Resolving Spontaneous Nonlinear Multi-Physics Flow Localisation in 3-D: Tackling Hardware Limit

Ludovic Räss[1,2], Samuel Omlin[2*], Yury Podladchikov[2]

[1] Stanford University, Department of Geophysics, Stanford CA, USA

[2] Swiss Geocomputing Centre, University of Lausanne, Lausanne, Switzerland

*now at CSCS - Swiss National Supercomputing Centre, Lugano, Switzerland

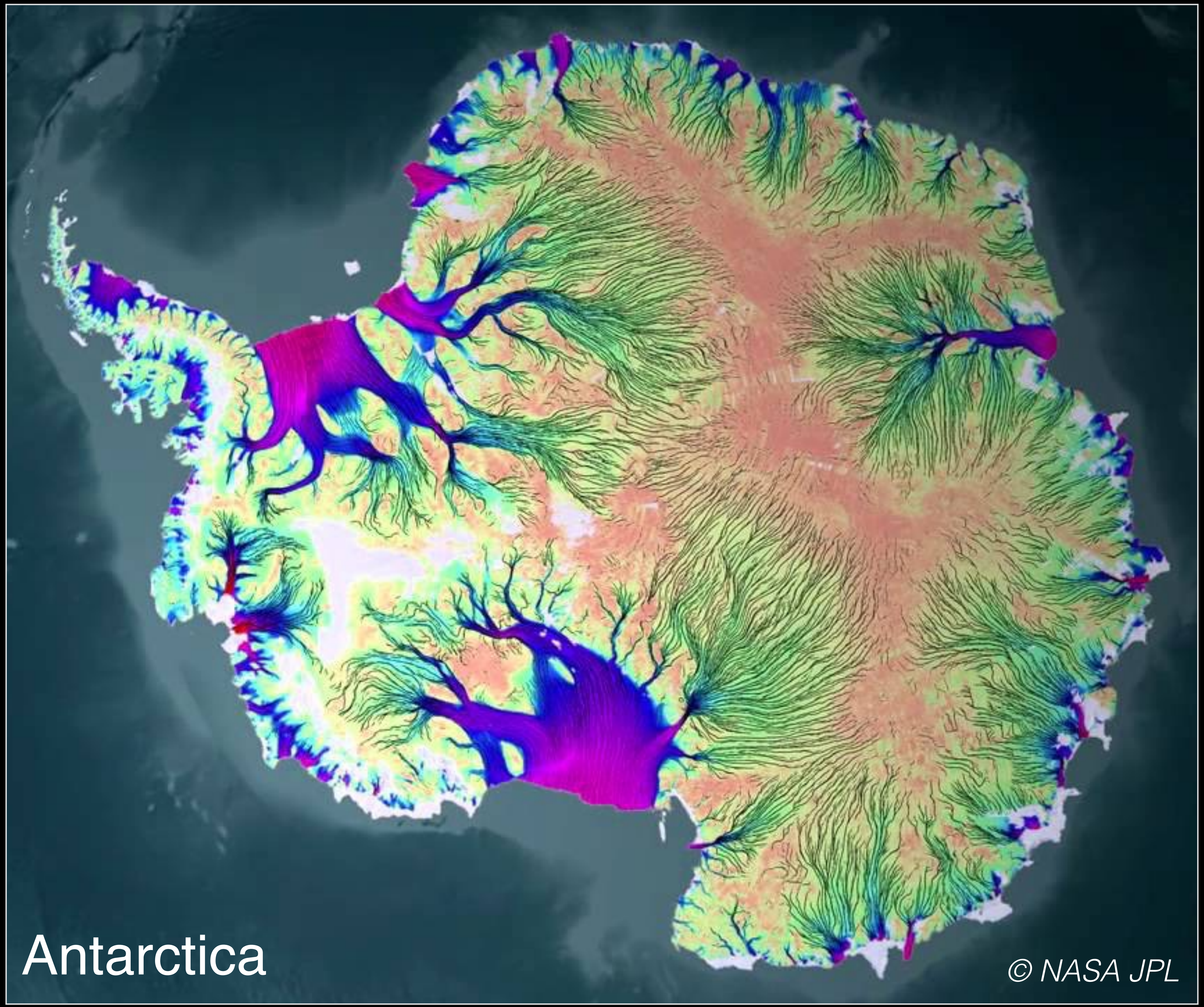
S9368 | 20 March 2019 | Nvidia GTC | Silicon Valley

UNIL | Université de Lausanne
Swiss Geocomputing Centre

FNSNF
Fonds national suisse
Schweizerischer Nationalfonds
Fondo nazionale svizzero
Swiss National Science Foundation

SIGMA

Multi-physics flow localisation



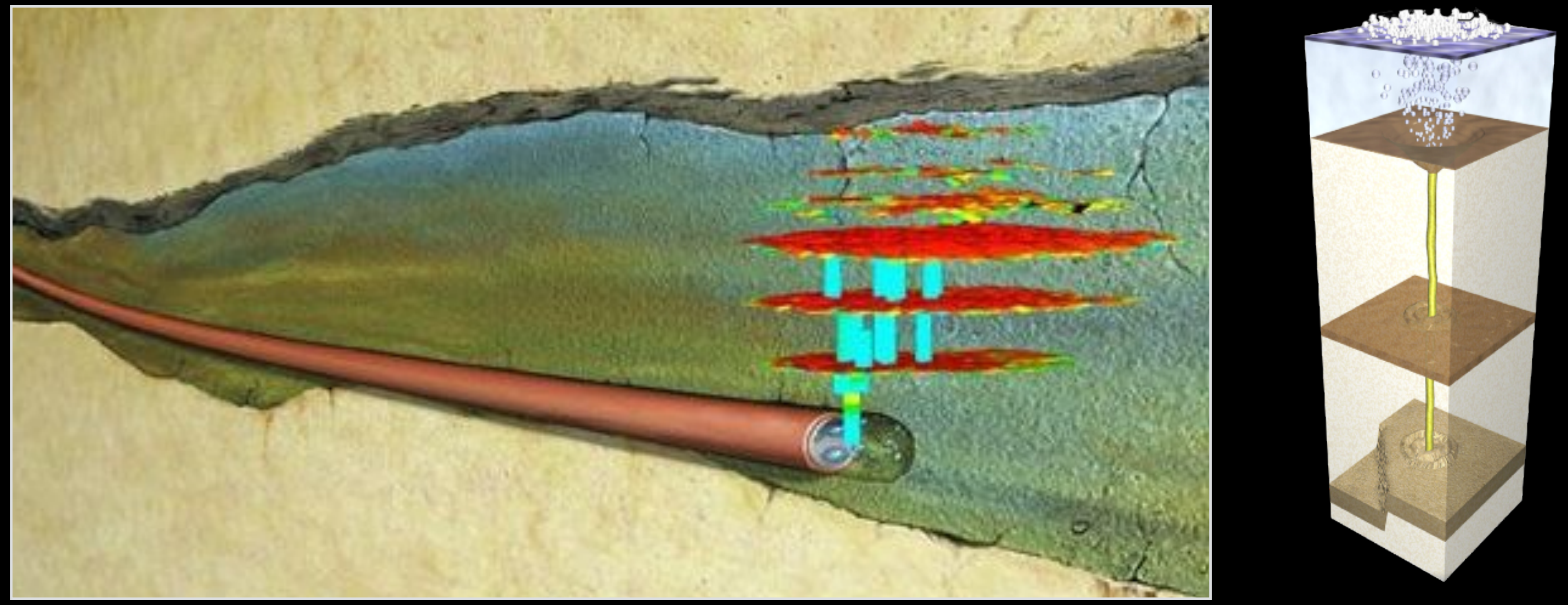

Subsurface reservoirs
Extraction and storage


Subsurface reservoirs
Extraction and storage
Pockmark
INT
H50
Truncations
H60
H70
Chimney
Løseth et al., 2011
Plaza-Faverola et al., 2011
© Statoil

Why do we care

Fast flowing ice streams

> Impact sea-level rise



High-permeability chimneys

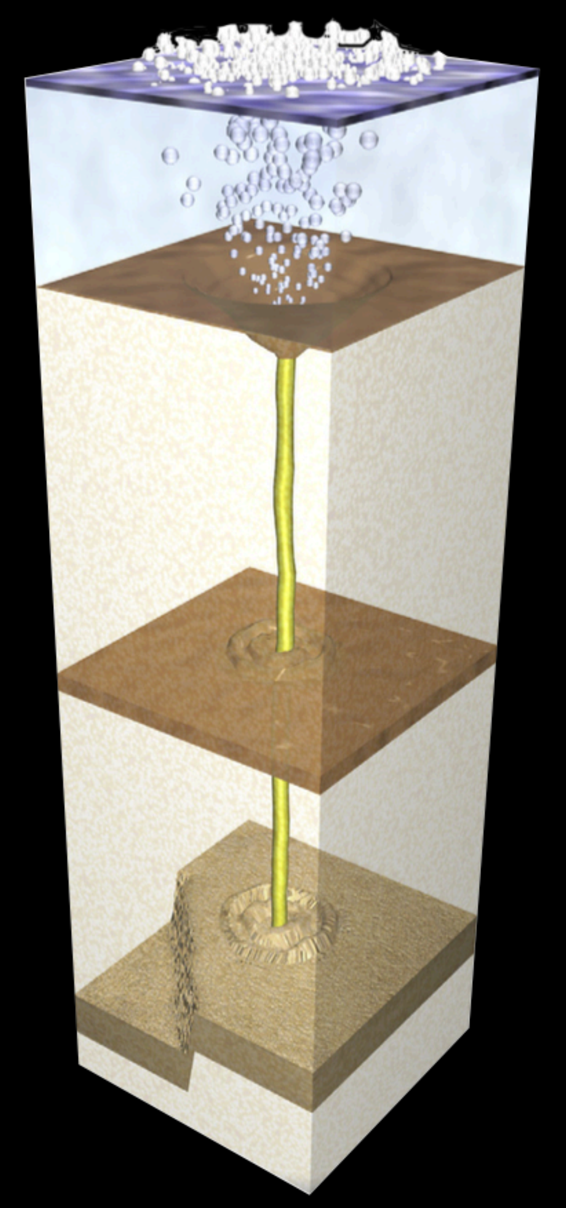
> Compromise safe CO_2 storage

- Macro-scale “reservoirs” | Micro-scale processes
- Multi-physics flow localisation trigger meso-scale instabilities
- Resolve the key physical processes to make accurate predictions

The challenge

Resolve meso-scale dynamics

- Develop predictive models to resolve nonlinear multi-physics processes
- Capture spontaneous localisation in space + time
- Large domains but highly localised action
- Computationally challenging:



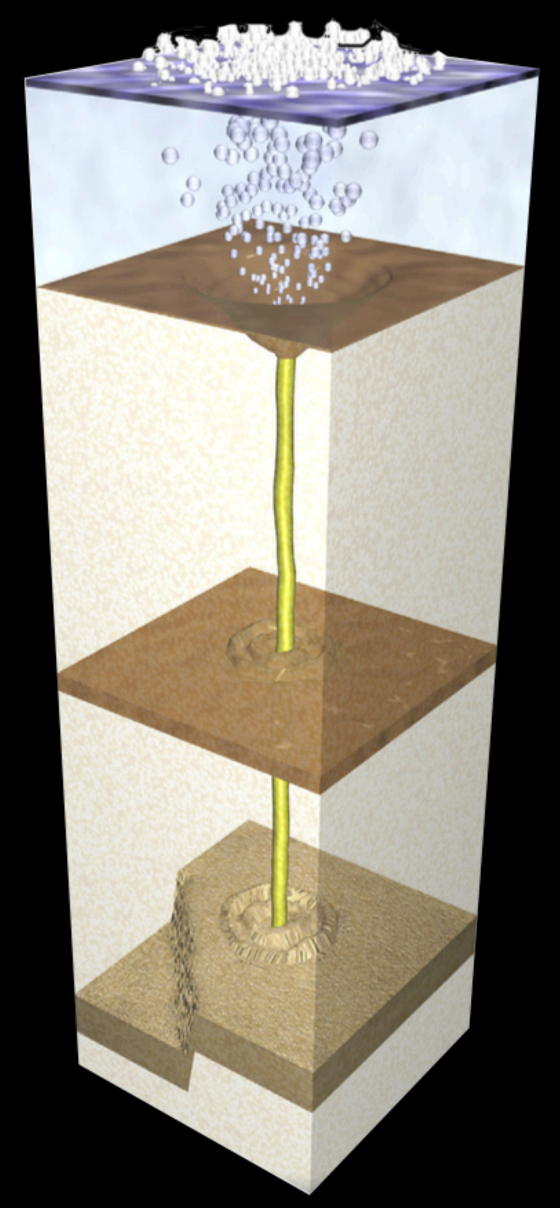
The challenge

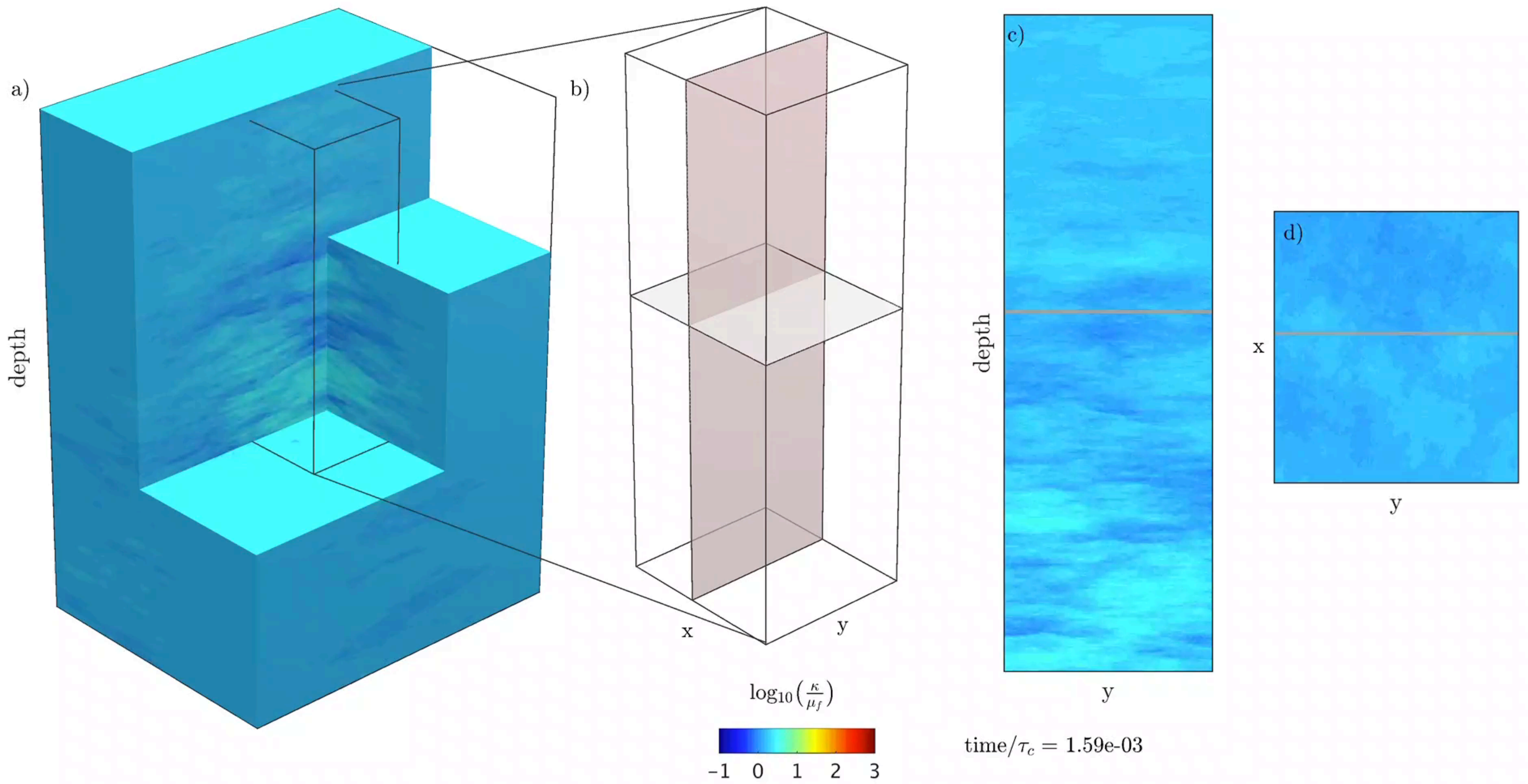
Resolve meso-scale dynamics

- Develop predictive models to resolve nonlinear multi-physics processes
- Capture spontaneous localisation in space + time
- Large domains but highly localised action
- Computationally challenging:

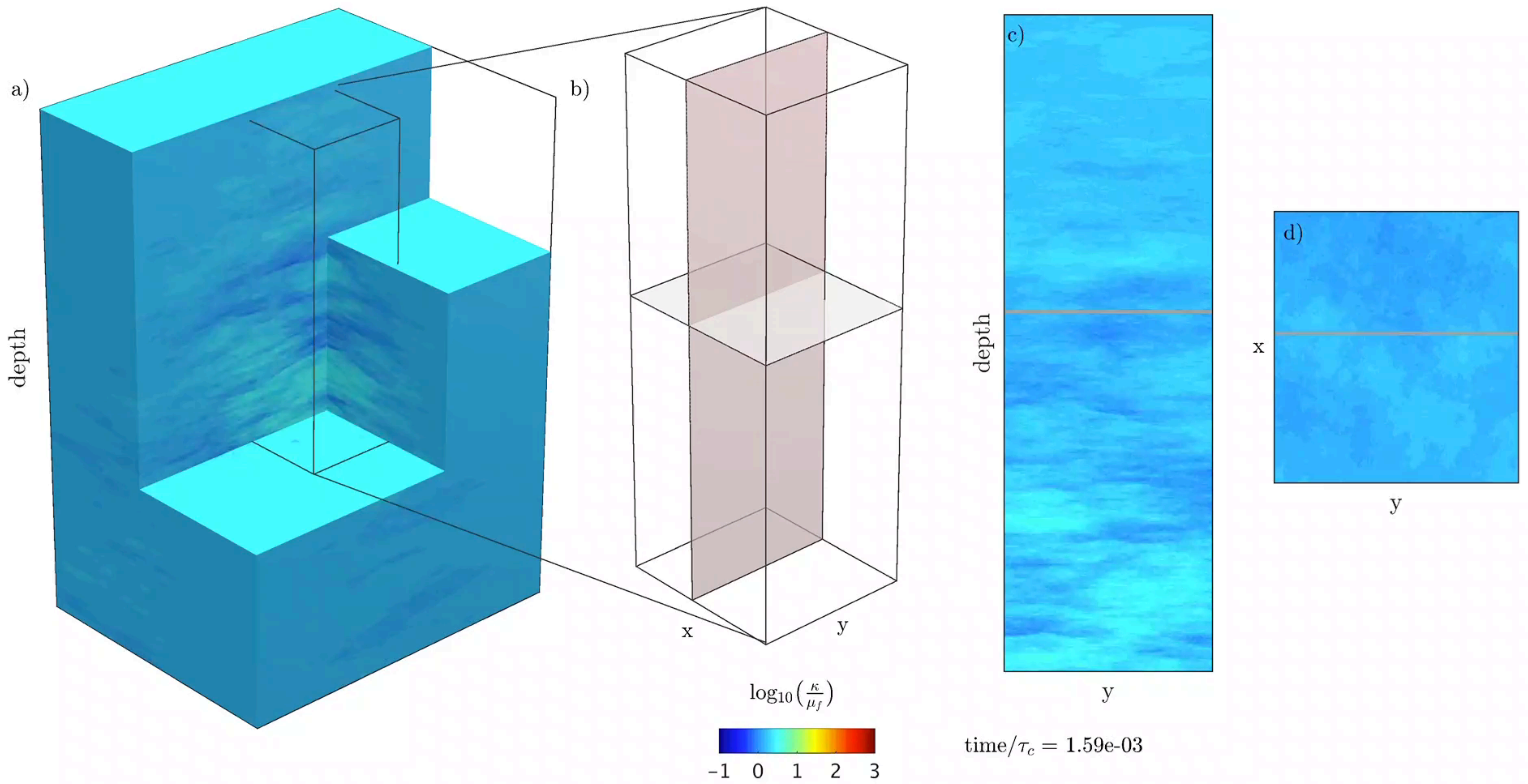
Extremely high-resolution 3-D forward models are mandatory

Requires a supercomputing approach





Räss et al., 2018. Nature Scientific Reports



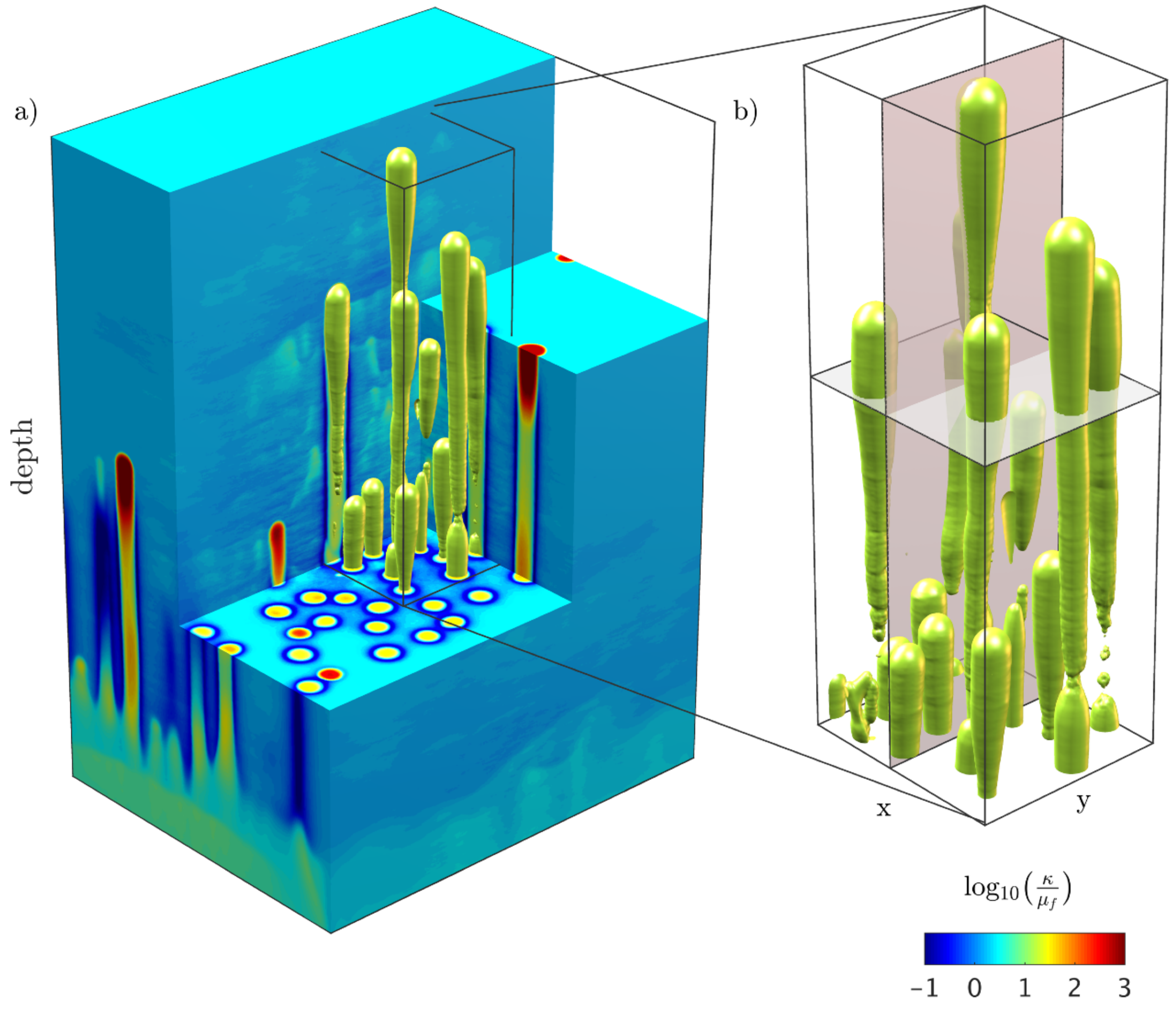
Räss et al., 2018. Nature Scientific Reports

Supercomputing in Earth Sciences

Resolution:

- space: 1000 x 1000 x 2000 grid points in 3-D
- time: 20’000 implicit time steps

- Finite-Difference method + staggered grid
- CUDA C + MPI
- 128 Titan Xm GPUs [home built supercomputer]
- ~600 GB memory footprint | 20 TB / 100 disk saves



Räss et al., 2018. Nature Scientific Reports

Supercomputing in Earth Sciences

SCIENTIFIC REPORTS

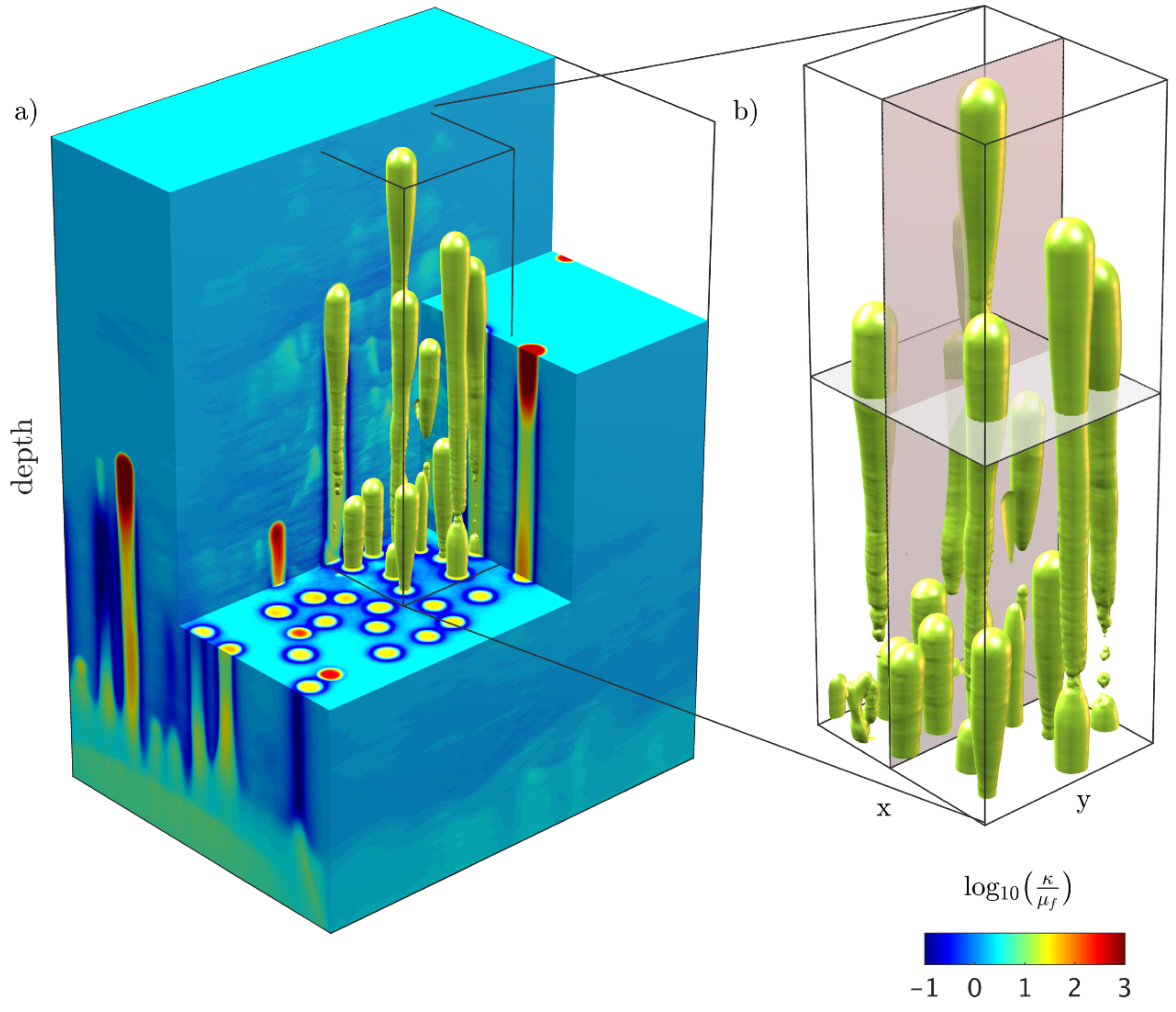
OPEN

Spontaneous formation of fluid escape pipes from subsurface reservoirs

Received: 29 March 2018

Ludovic Räss[1,2], Nina S. C. Simon[3] & Yury Y. Podladchikov[1,2]

- CUDA C + MPI
- 128 Titan Xm GPUs [home built supercomputer]
- ~600 GB memory footprint | 20 TB / 100 disk saves

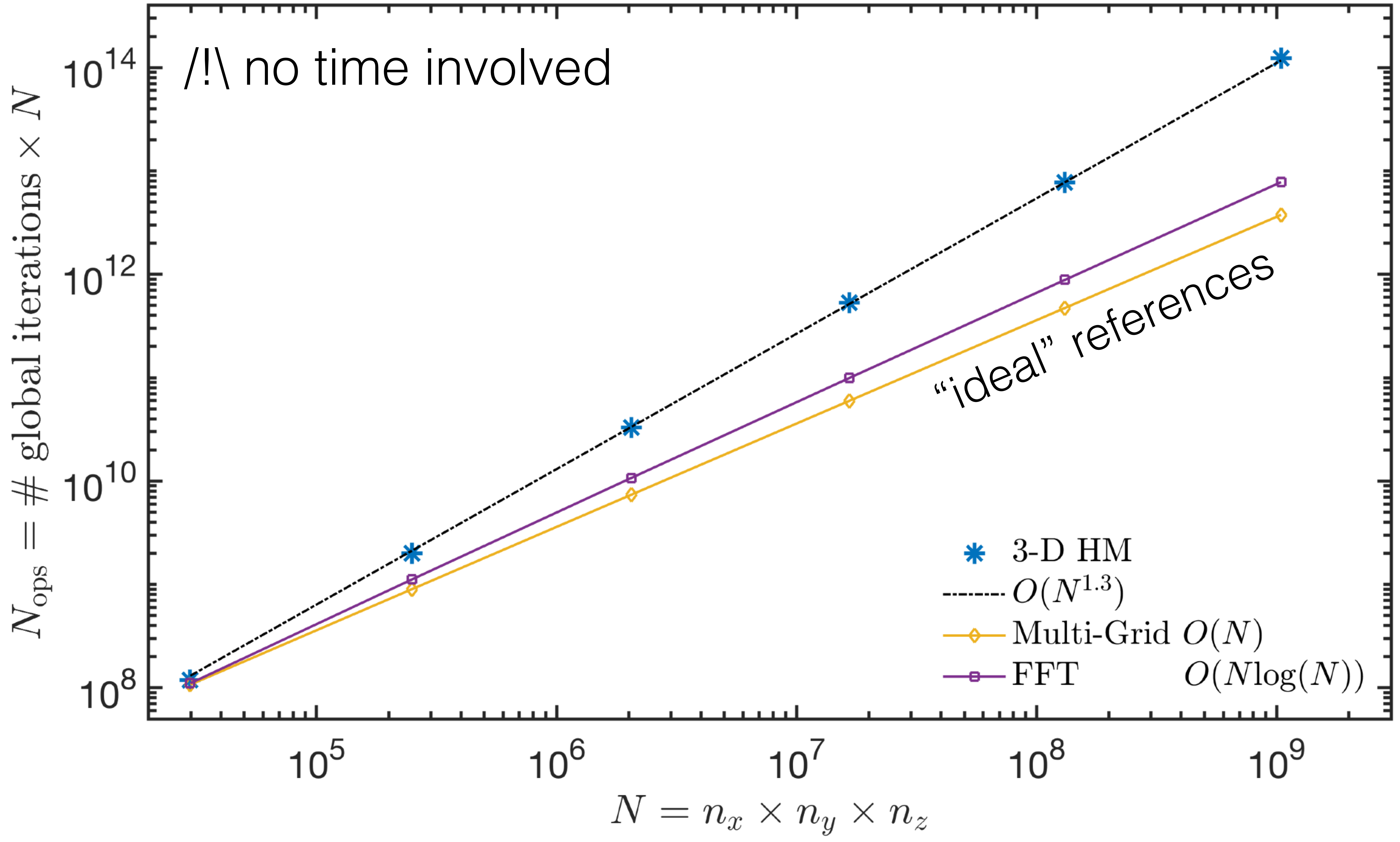


Runs close to hardware limit !

Räss et al., 2018. Nature Scientific Reports

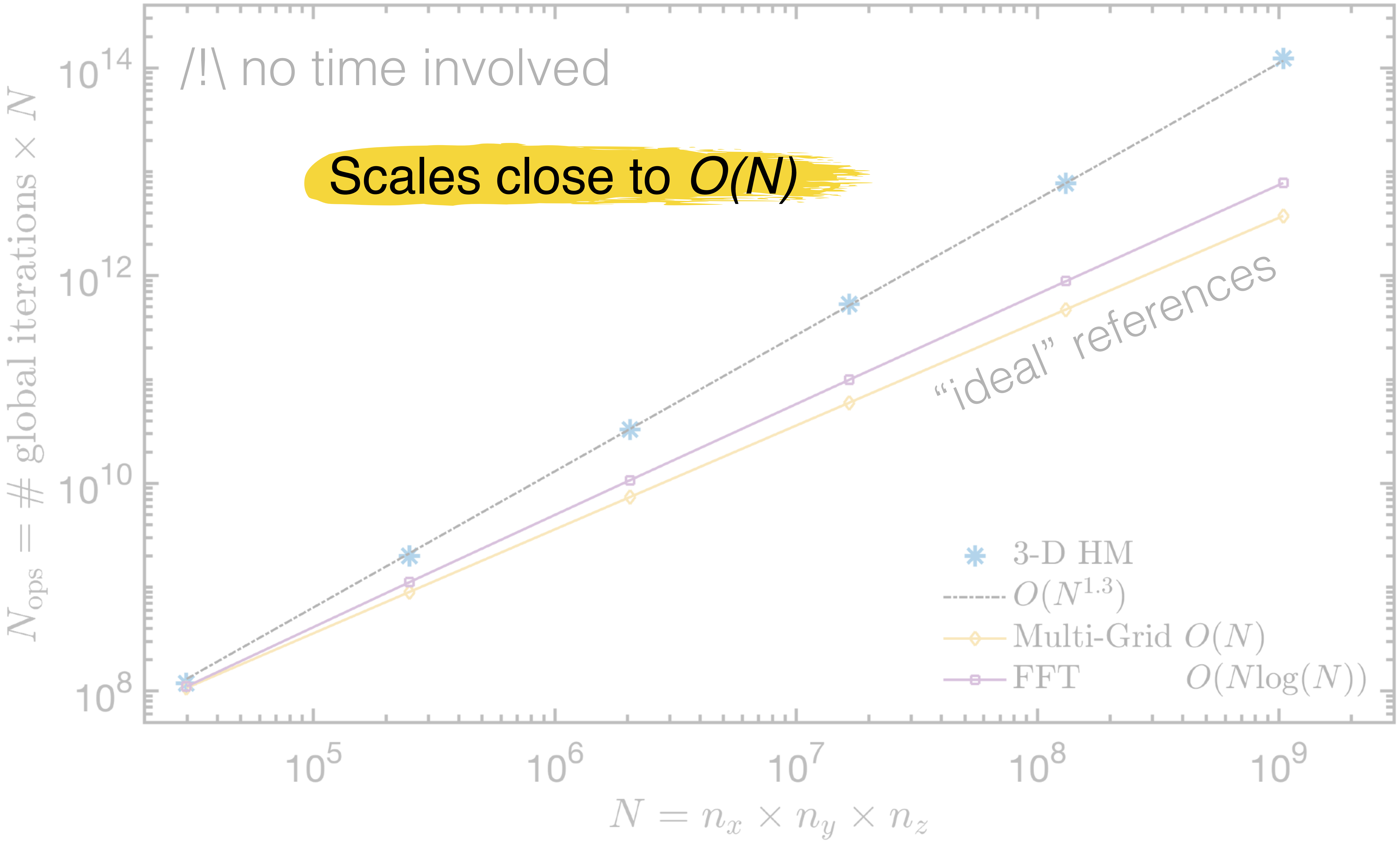
1/ Pseudo-Transient solver

Scaling of our iterative GPU solver with numerical grid resolution N



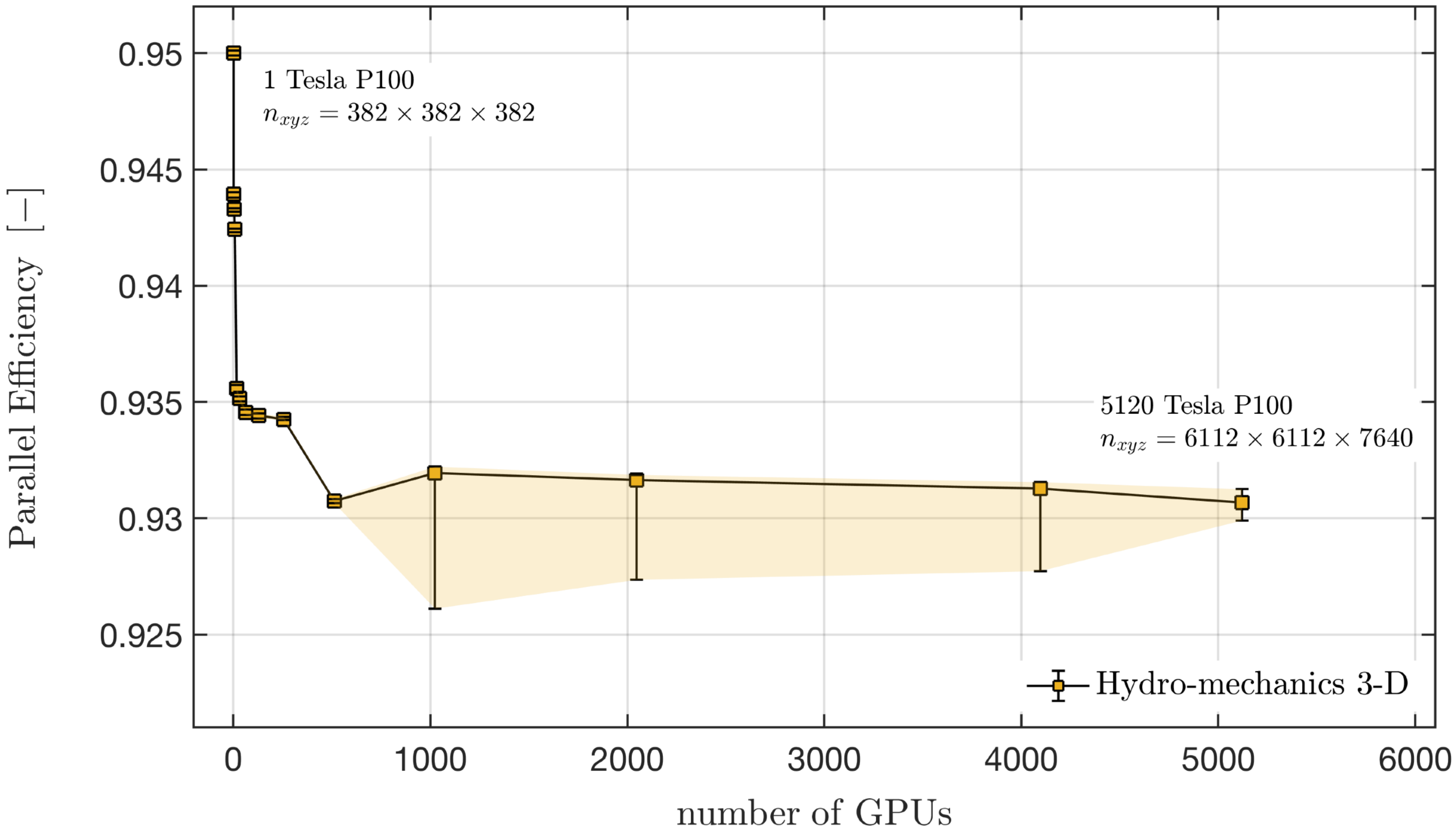
1/ Pseudo-Transient solver

Scaling of our iterative GPU solver with numerical grid resolution N



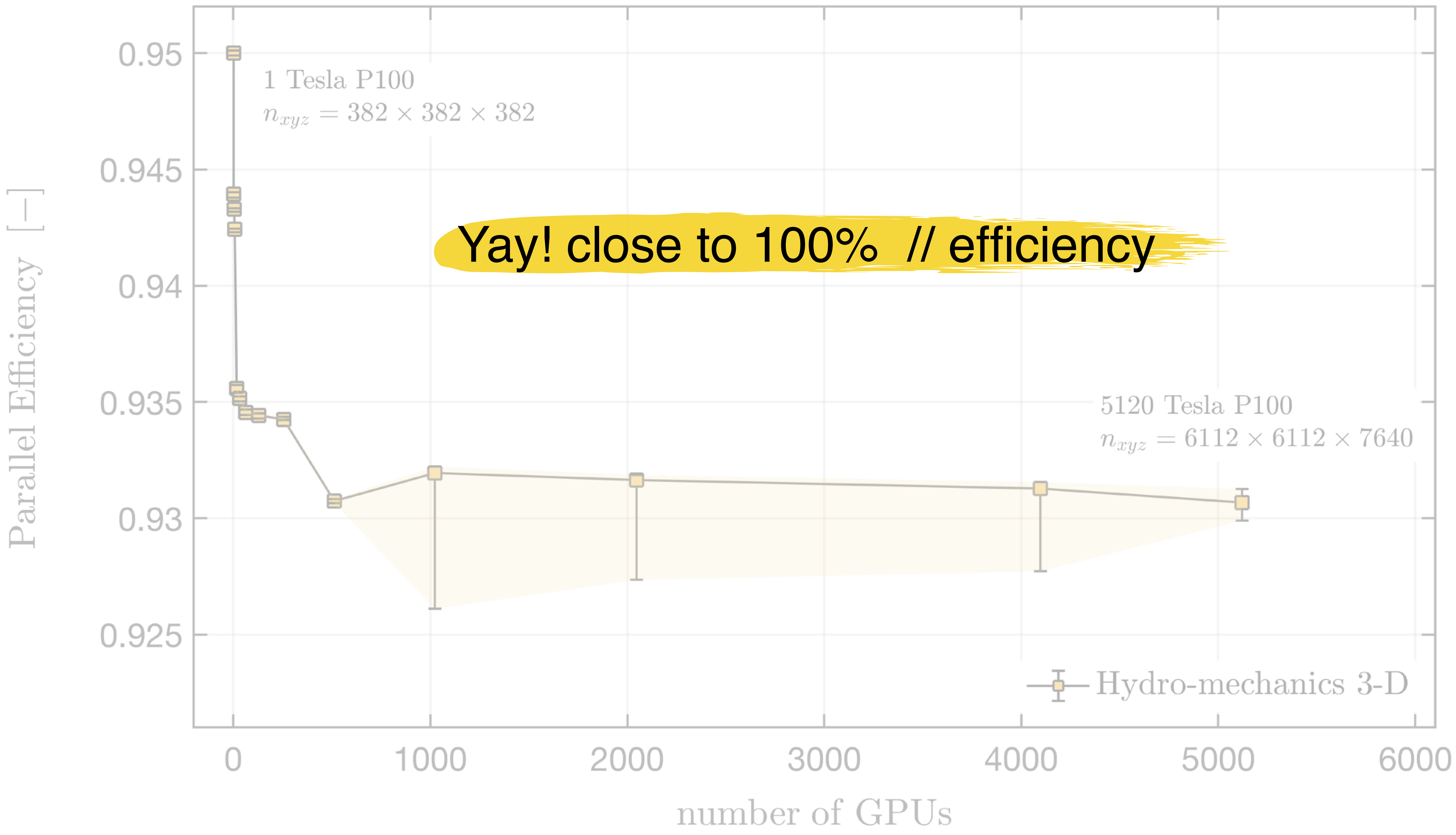
2/ Hide MPI communication

Weak scaling on Cray XC50 “Piz Daint” @ CSCS | **5120 Nvidia P100 GPUs**



2/ Hide MPI communication

Weak scaling on Cray XC50 “Piz Daint” @ CSCS | **5120 Nvidia P100 GPUs**



The road to a [fast] solution

Agenda

- Results ✓
- 1/ Pseudo-Transient - *be close to physics*
- 2/ Parallel efficiency - *hide MPI communication*
- Summary & Outlook
- Q&A

Solve 3-D solid / fluid mechanics

- Solve 3-D nonlinear hydro-mechanics PDEs | 15 mandatory fields to update

$$0 = \nabla_j(\bar{\tau}_{ij} - \bar{p}\delta_{ij}) - \bar{\rho}g_i$$

$$0 = \nabla_k v_k + \frac{\bar{p} - p^f}{\eta_\phi(1-\phi)}$$

$$0 = \nabla_k q_k^D - \frac{\bar{p} - p^f}{\eta_\phi(1-\phi)}$$

$$\frac{\partial \phi}{\partial t} = (1-\phi)\nabla_k v_k$$

$$\frac{\partial \bar{\tau}_{ij}}{\partial t} = f(v, G, \mu_s)$$

$$\eta_\phi = f(\phi, \bar{p}, p^f, \dot{\epsilon}_{II})$$

$$\mu = f(\phi, \dot{\epsilon}_{II})$$

$$k_\phi = k_0(\phi/\phi_0)^3$$

Utilise a fully local physics-based iterative method: **Pseudo-Transient**

Solve 3-D solid / fluid mechanics

- Solve 3-D nonlinear hydro-mechanics PDEs | 15 mandatory fields to update

$$0 = \nabla_j(\bar{\tau}_{ij} - \bar{p}\delta_{ij}) - \bar{\rho} g_i$$

$$0 = \nabla_k v_k + \frac{\bar{p} - p^f}{\eta_\phi(1-\phi)}$$

$$0 = \nabla_k q_k^D - \frac{\bar{p} - p^f}{\eta_\phi(1-\phi)}$$

$$\frac{\partial \phi}{\partial t} = (1-\phi)\nabla_k v_k$$

$$\frac{\partial \bar{\tau}_{ij}}{\partial t} = f(v, G, \mu_s)$$

$$\eta_\phi = f(\phi, \bar{p}, p^f, \dot{\epsilon}_{II})$$

$$\mu = f(\phi, \dot{\epsilon}_{II})$$

$$k_\phi = k_0(\phi/\phi_0)^3$$

Utilise a fully local physics-based iterative method: **Pseudo-Transient**

- Avoid being “killed” by global communication & memory footprint
- Exploit the low level parallelism of GPUs + available supercomputing power

1/ Pseudo-Transient iterations

A general approach

- e.g. solution to an elliptic problem:

$$C = \frac{\partial^2 A}{\partial x^2}$$

$$0 = \frac{\partial^2 A}{\partial x^2} - C = f_{\mathrm{A}}$$

1/ Pseudo-Transient iterations

A general approach

- e.g. solution to an elliptic problem:

$$C = \frac{\partial^2 A}{\partial x^2}$$

$$0 = \frac{\partial^2 A}{\partial x^2} - C = f_\mathrm{A}$$

- Naive iterations (1st order):

$$\frac{\partial A}{\partial \tau_\mathrm{A}} = f_\mathrm{A}$$

$$A^{[k+1]} = A^{[k]} + \Delta\tau_\mathrm{A} f_\mathrm{A}^{[k]}$$

1/ Pseudo-Transient iterations

A general approach

- e.g. solution to an elliptic problem:

$$C = \frac{\partial^2 A}{\partial x^2}$$

$$0 = \frac{\partial^2 A}{\partial x^2} - C = f_\mathrm{A}$$

- Naive iterations (1st order):

$$\cancelto{0}{\frac{\partial A}{\partial \tau_\mathrm{A}}} = f_\mathrm{A}$$

$$A^{[k+1]} = A^{[k]} + \Delta\tau_\mathrm{A} \cancelto{0}{f_\mathrm{A}^{[k]}}$$

1/ Pseudo-Transient iterations

A general approach

- e.g. solution to an elliptic problem: $$C = \frac{\partial^2 A}{\partial x^2}$$

$$0 = \frac{\partial^2 A}{\partial x^2} - C = f_{\mathrm{A}}$$

- Naive iterations (1st order): $$\cancel{\frac{\partial A}{\partial \tau_{\mathrm{A}}}}^{\,0} = f_{\mathrm{A}}$$

$$A^{[k+1]} = A^{[k]} + \Delta\tau_{\mathrm{A}} \cancel{f_{\mathrm{A}}^{[k]}}^{\,0}$$

(pseudo) time step

1/ Pseudo-Transient iterations

A general approach

- e.g. solution to an elliptic problem:

$$C = \frac{\partial^2 A}{\partial x^2}$$

$$0 = \frac{\partial^2 A}{\partial x^2} - C = f_{\mathrm{A}}$$

$$\begin{aligned} f_{\mathrm{v}} &= \nabla_j(\bar{\tau}_{ij} - \bar{p}\delta_{ij}) - \bar{\rho} g_i \\ f_{\bar{\mathrm{p}}} &= \nabla_k v_k + \frac{\bar{p} - p^f}{\eta_\phi(1-\phi)} \\ f_{\mathrm{p^f}} &= \nabla_k q_k^D - \frac{\bar{p} - p^f}{\eta_\phi(1-\phi)} \end{aligned}$$

- Naive iterations (1st order):

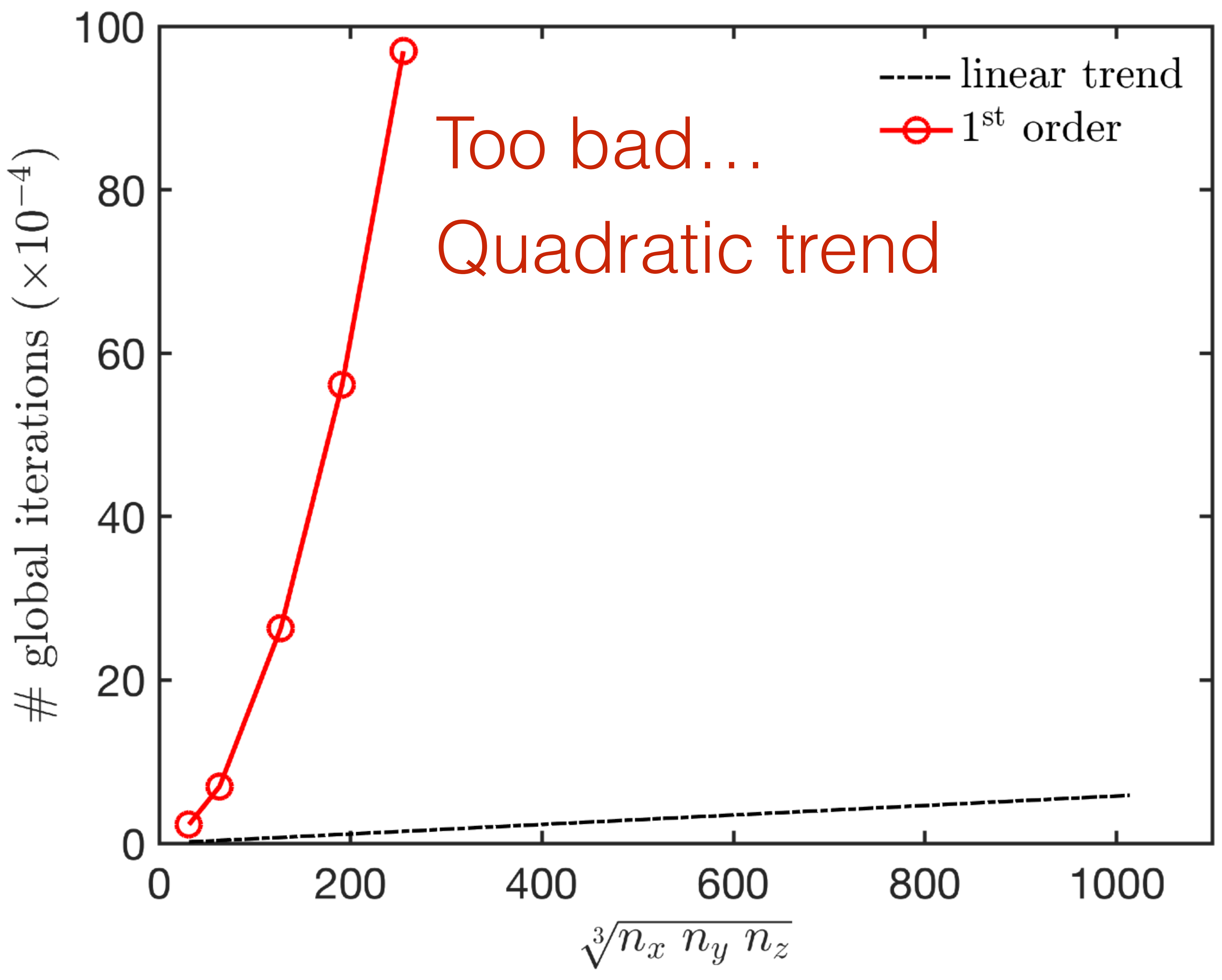
$$\cancel{\frac{\partial A}{\partial \tau_{\mathrm{A}}}}^{\,0} = f_{\mathrm{A}}$$

$$A^{[k+1]} = A^{[k]} + \Delta\tau_{\mathrm{A}} \cancel{f_{\mathrm{A}}^{[k]}}^{\,0}$$

(pseudo) time step

Pseudo-Transient | naive

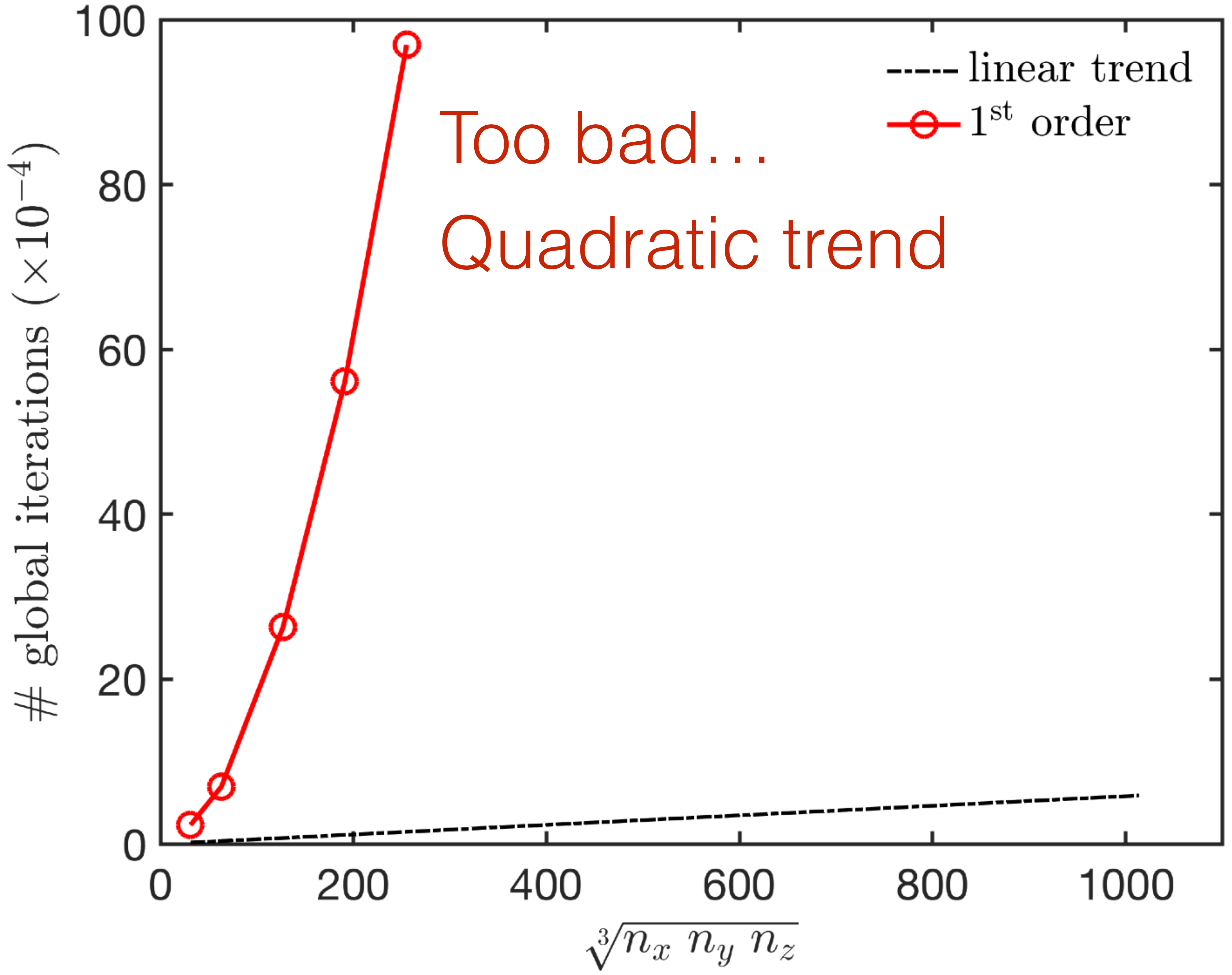
- 1st order scheme | 3-D hydro-mechanics



Pseudo-Transient | naive

- 1^{st} order scheme | 3-D hydro-mechanics

Simple - but does not scale with resolution increase



Pseudo-Transient iterations

An improved approach

- Second order iterations:

Frankel, 1950

$$\frac{\partial A}{\partial \tau_{\mathrm{A}}} = f_{\mathrm{A}}$$

$$A^{[k+1]} = A^{[k]} + \Delta\tau_{\mathrm{A}} \; f_{\mathrm{A}}^{[k]}$$

Pseudo-Transient iterations

An improved approach

- Second order iterations:

Frankel, 1950

$$\alpha \frac{\partial^2 A}{\partial \tau_A^2} + \frac{\partial A}{\partial \tau_A} = f_A$$

$$A^{[k+1]} = A^{[k]} + \Delta\tau_A \left(f_A^{[k]} + (1 - \nu/n_i)\, f_A^{[k-1]} \right)$$

Pseudo-Transient iterations

An improved approach

- Second order iterations:

Frankel, 1950

$$\alpha \frac{\partial^2 A}{\partial \tau_{\mathrm{A}}^2} + \frac{\partial A}{\partial \tau_{\mathrm{A}}} = f_{\mathrm{A}}$$

$$A^{[k+1]} = A^{[k]} + \Delta\tau_{\mathrm{A}} \left(f_{\mathrm{A}}^{[k]} + (1 - \nu/n_i)\, f_{\mathrm{A}}^{[k-1]} \right)$$ damping

Pseudo-Transient iterations

An improved approach

- Second order iterations:

Frankel, 1950

$$\alpha \frac{\partial^2 A}{\partial \tau_A^2} + \frac{\partial A}{\partial \tau_A} = f_A$$

$$A^{[k+1]} = A^{[k]} + \Delta\tau_A \left(f_A^{[k]} + (1 - \nu/n_i)\, f_A^{[k-1]} \right)$$

damping

0 0 0 0

Pseudo-Transient iterations

An improved approach

- Second order iterations:

Frankel, 1950

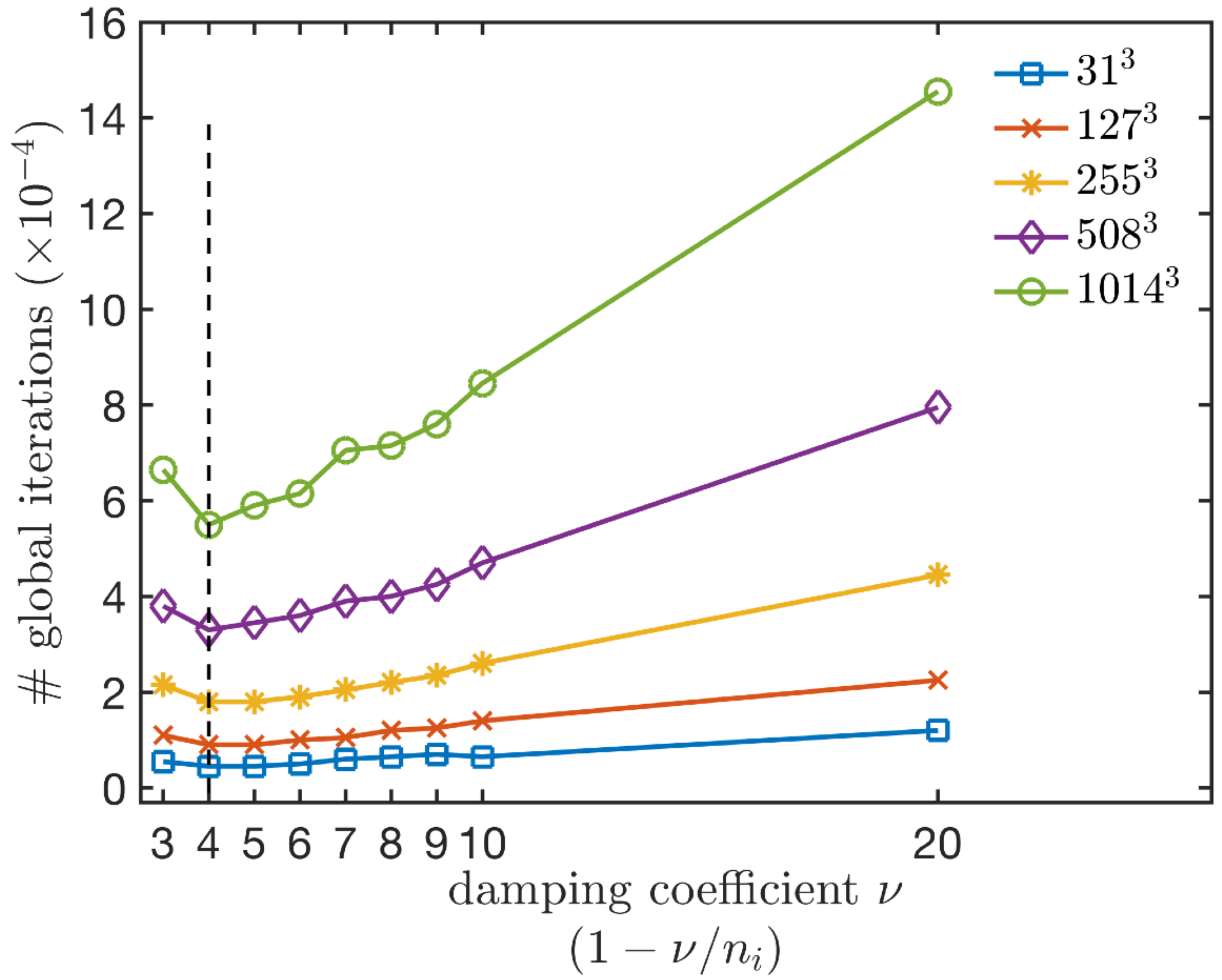
$$\alpha \frac{\partial^2 A}{\partial \tau_A^2} + \frac{\partial A}{\partial \tau_A} = f_A$$

0 0

$$A^{[k+1]} = A^{[k]} + \Delta\tau_A \left(f_A^{[k]} + (1 - \nu/n_i)\, f_A^{[k-1]} \right)$$

0 0

damping



free parameter ν is resolution independent

Pseudo-Transient iterations

An improved approach

- Second order iterations: $$\alpha \cancelto{0}{\frac{\partial^2 A}{\partial \tau_{\mathrm{A}}^2}} + \cancelto{0}{\frac{\partial A}{\partial \tau_{\mathrm{A}}}} = f_{\mathrm{A}}$$

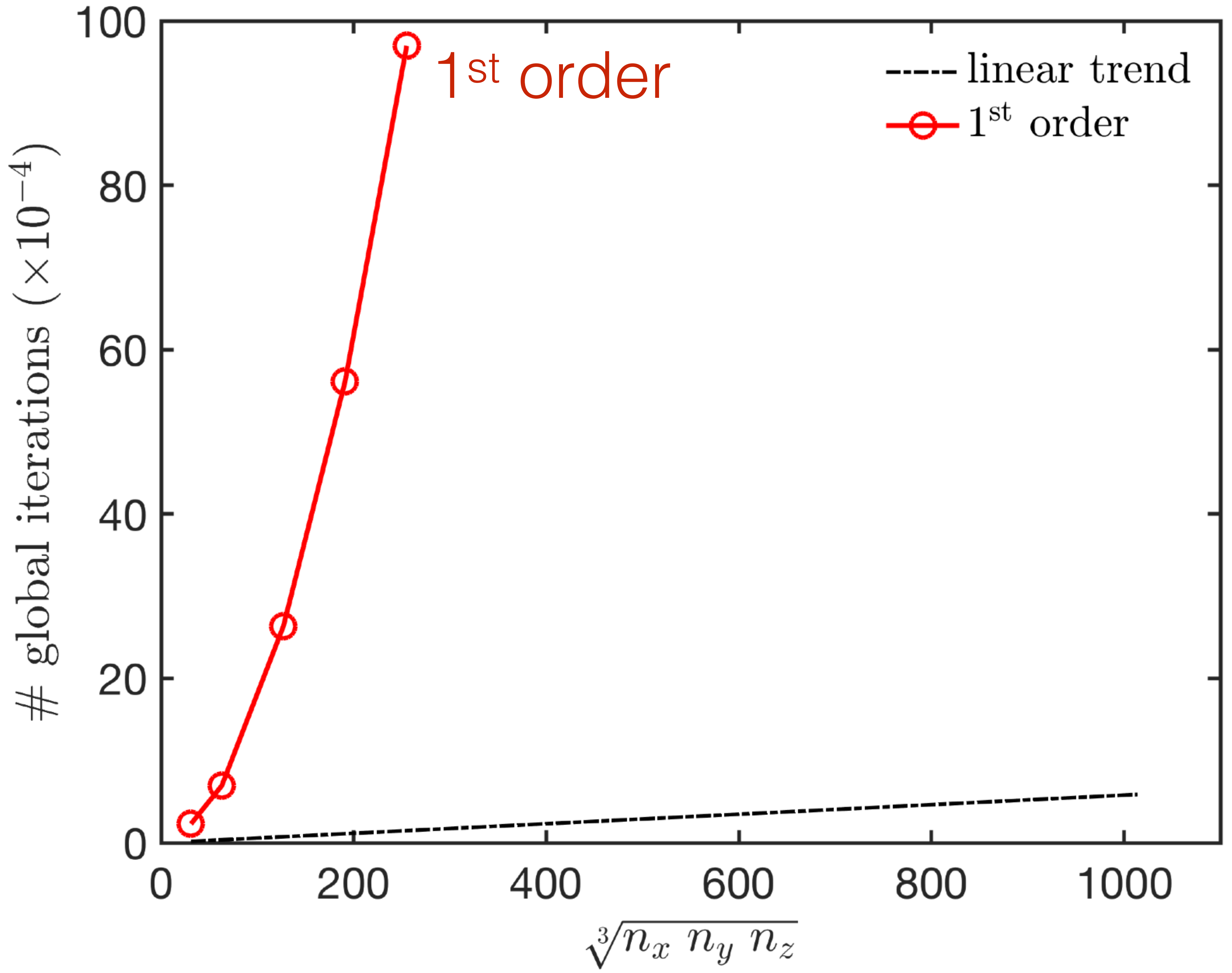
  Frankel, 1950

$$A^{[k+1]} = A^{[k]} + \Delta\tau_{\mathrm{A}} \left(\cancelto{0}{f_{\mathrm{A}}^{[k]}} + (1 - \nu/n_i)\, \cancelto{0}{f_{\mathrm{A}}^{[k-1]}} \right)$$ damping

- No global reduction: $\Delta\tau_{\mathrm{A}}$ is local (diagonal preconditionner)
- Only local communication: like propagation of physical information
- Same workflow on each grid point: perfect for GPUs !

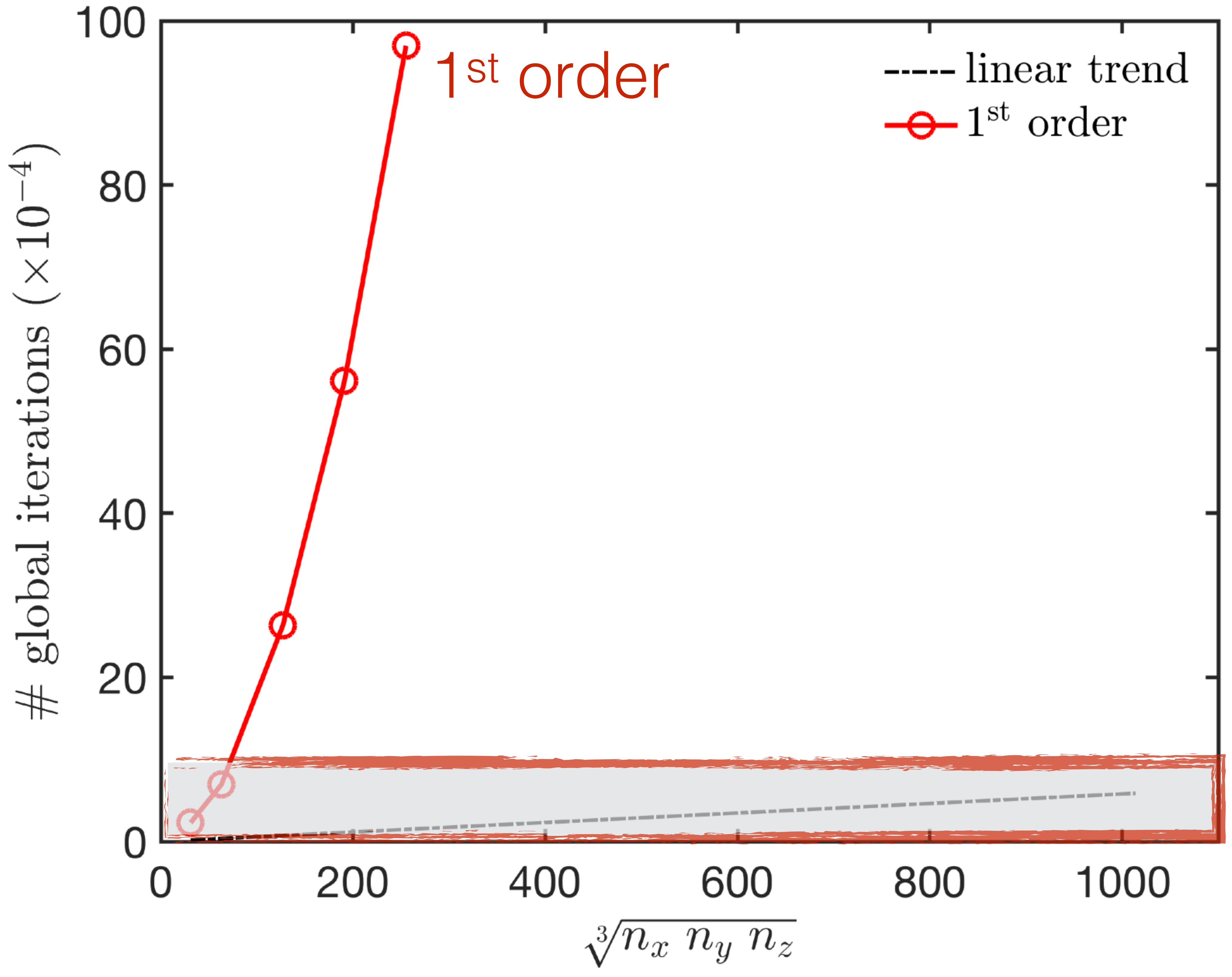
Pseudo-Transient | improved

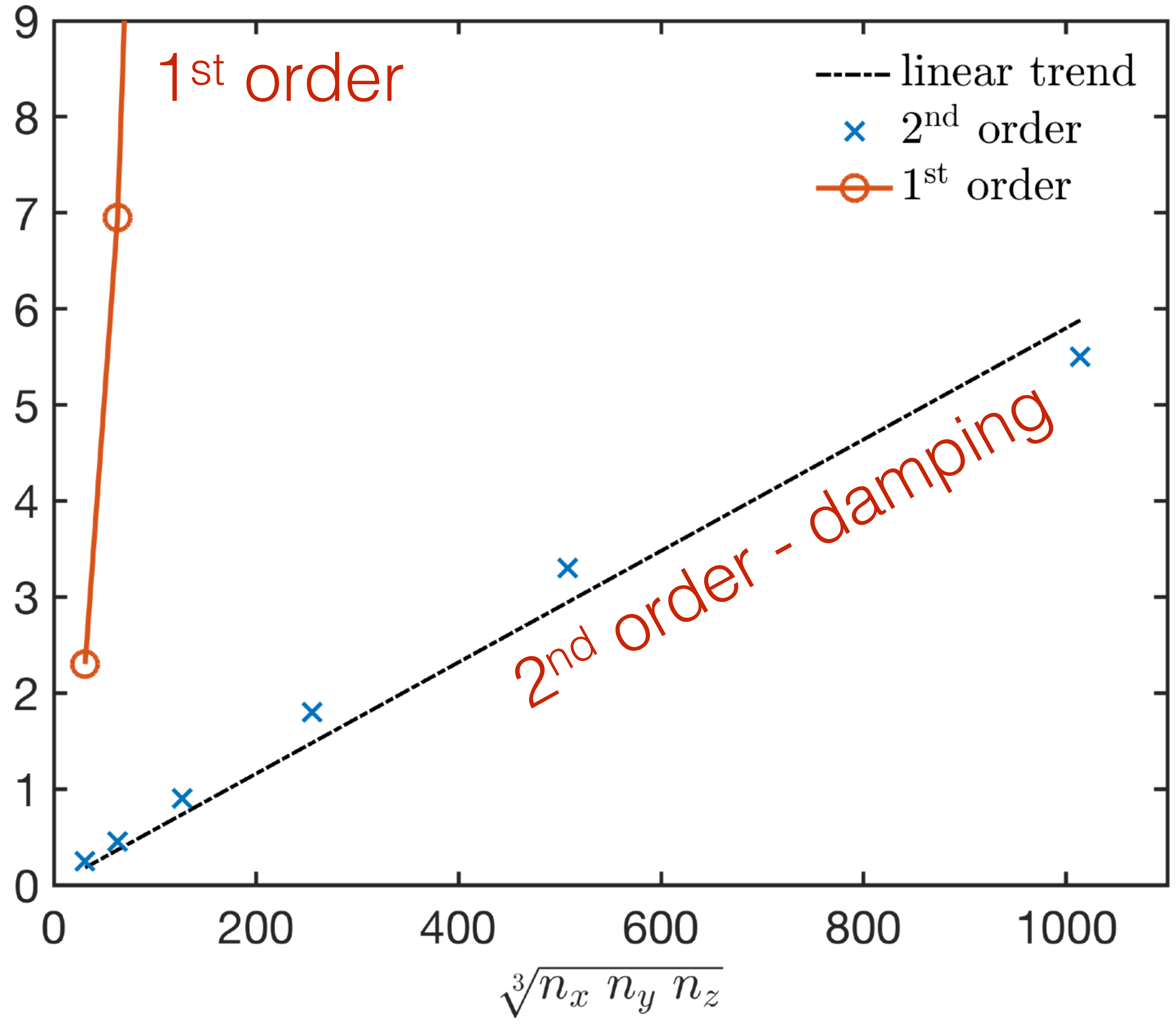
- 1^{st} versus 2^{nd} order scheme | 3-D hydro-mechanics



Pseudo-Transient | improved

- 1st versus 2nd order scheme | 3-D hydro-mechanics

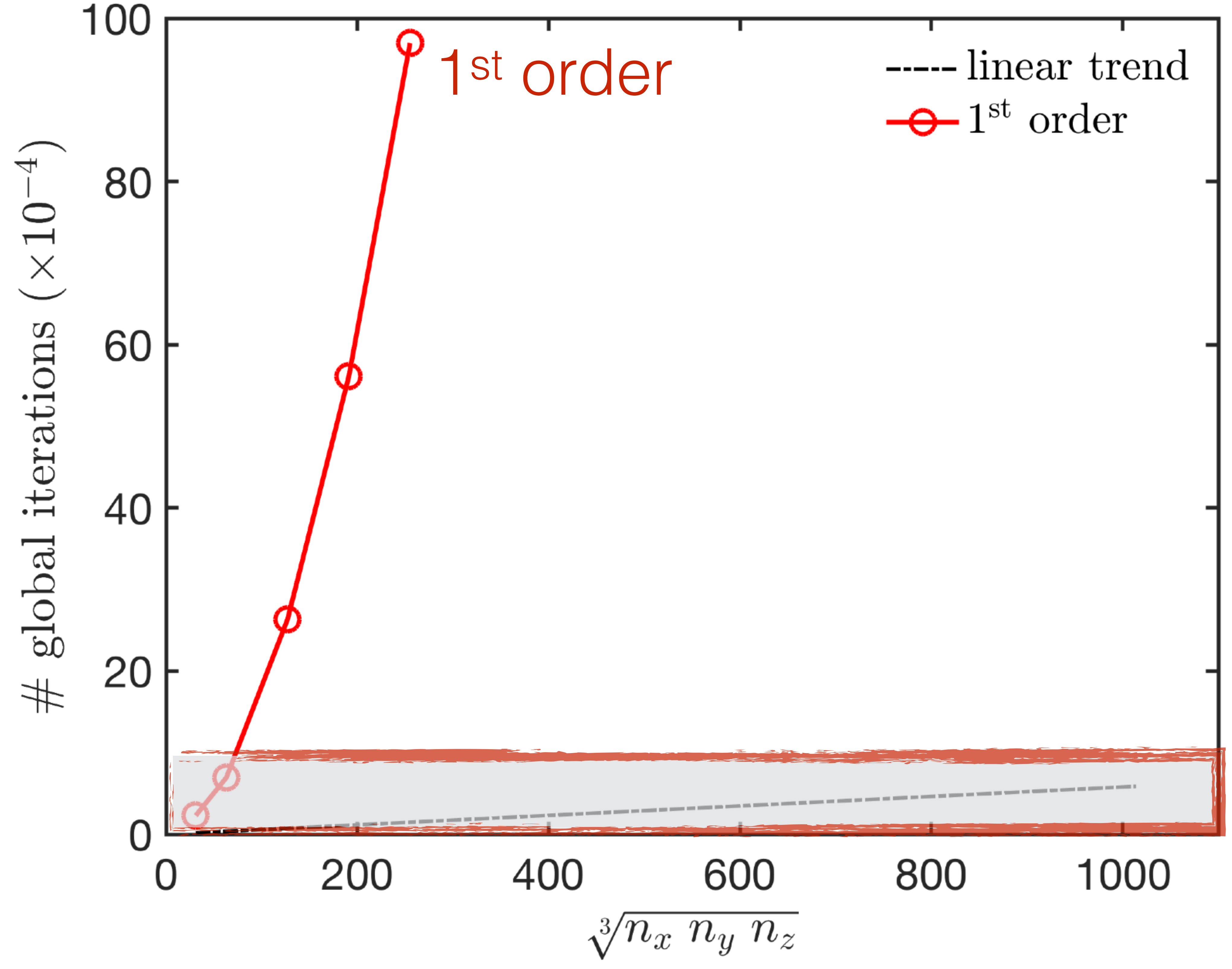


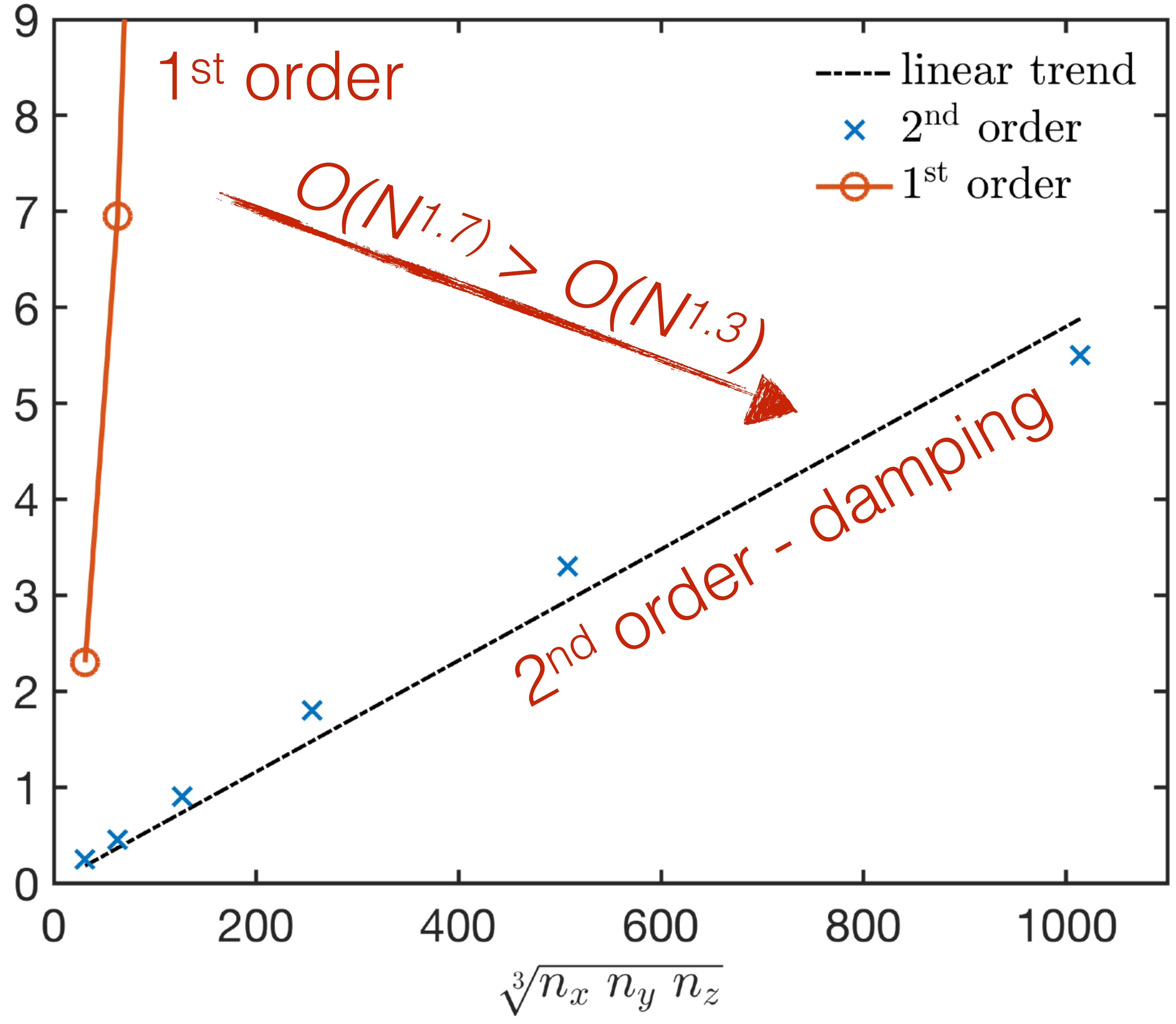


Pseudo-Transient | improved

- 1st versus 2nd order scheme | 3-D hydro-mechanics

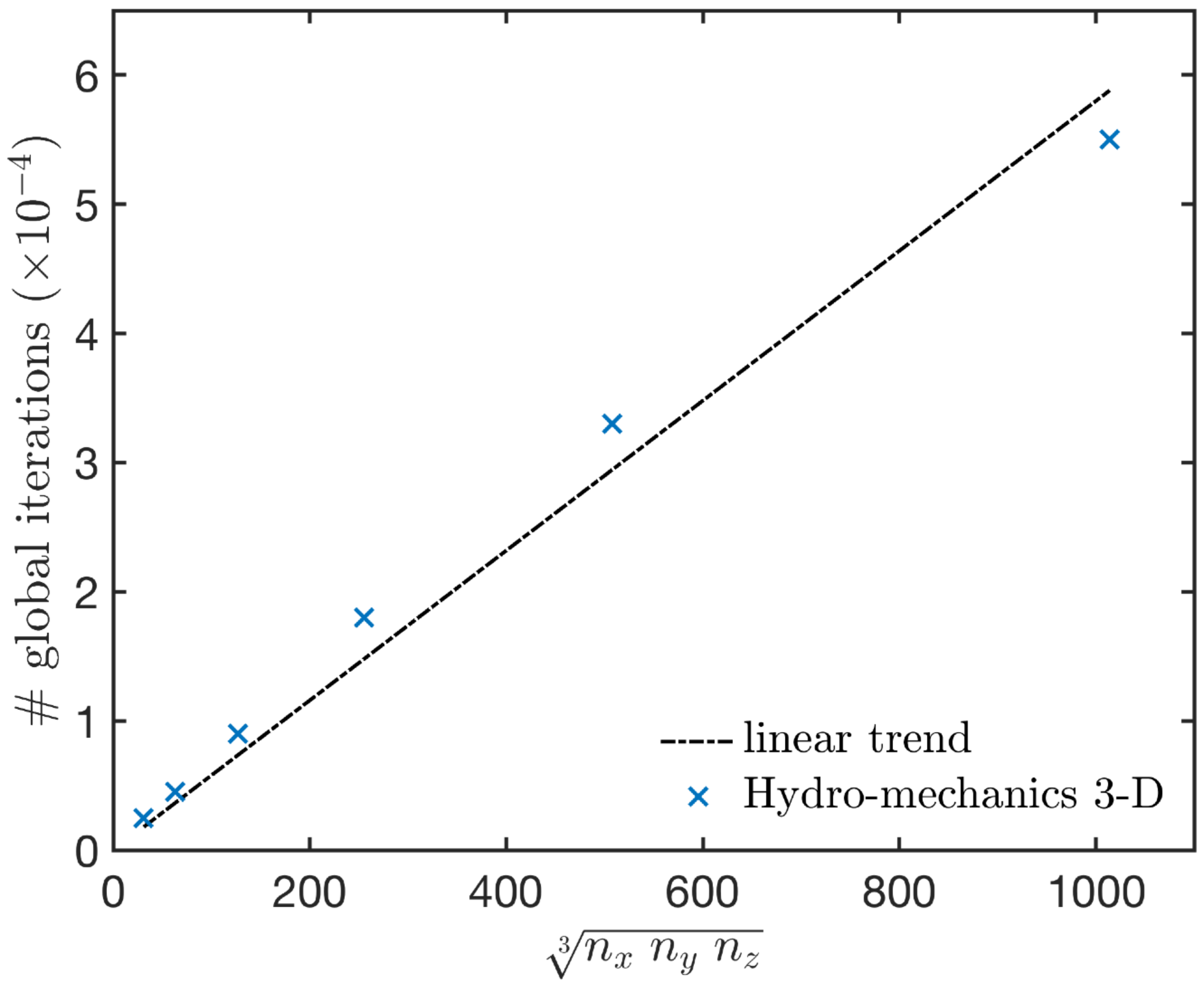
Yay! Scales $O(N^{1.3})$





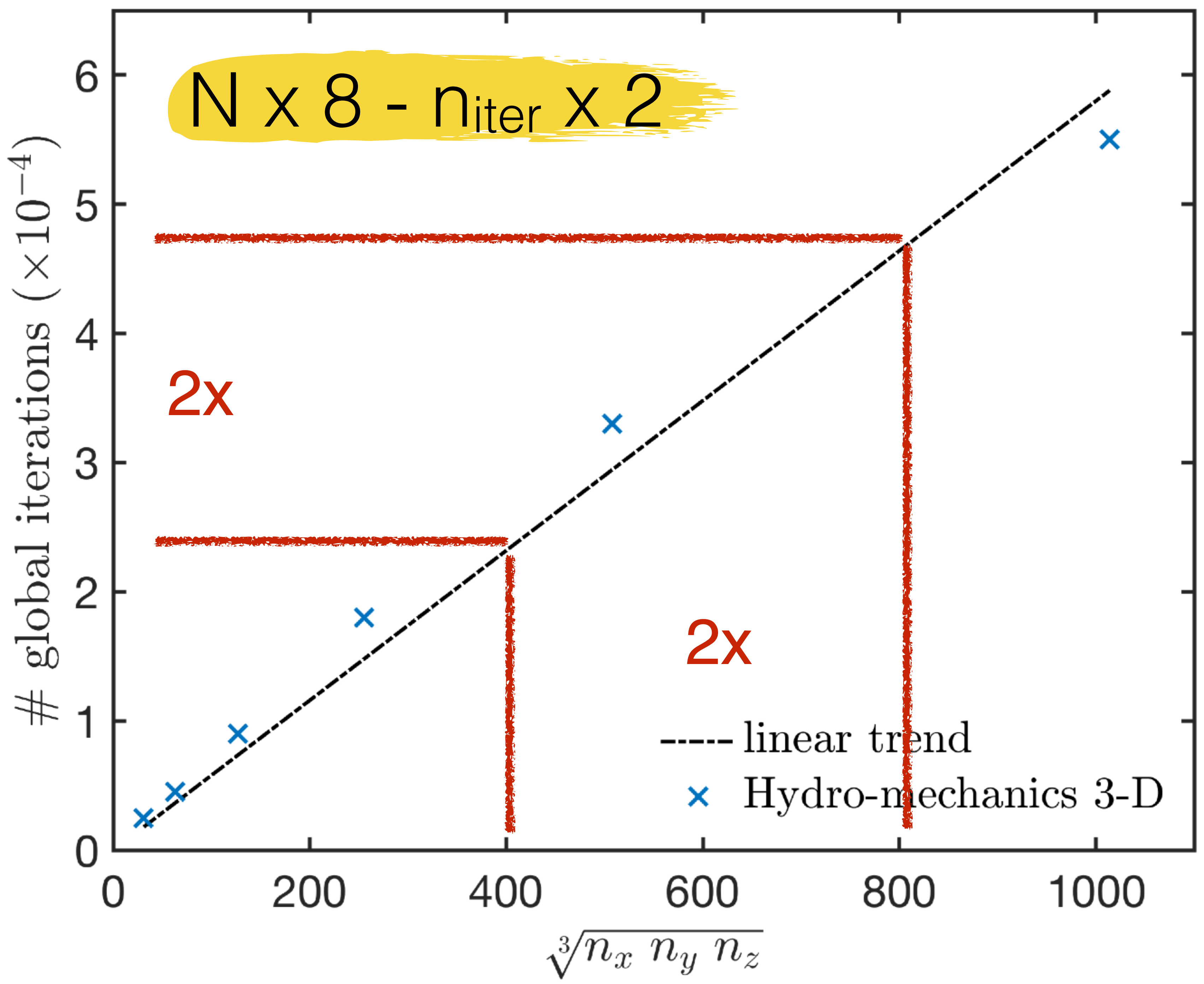
Iteration count ≠ time to solution

- Comparison to algorithms with ideal scalability



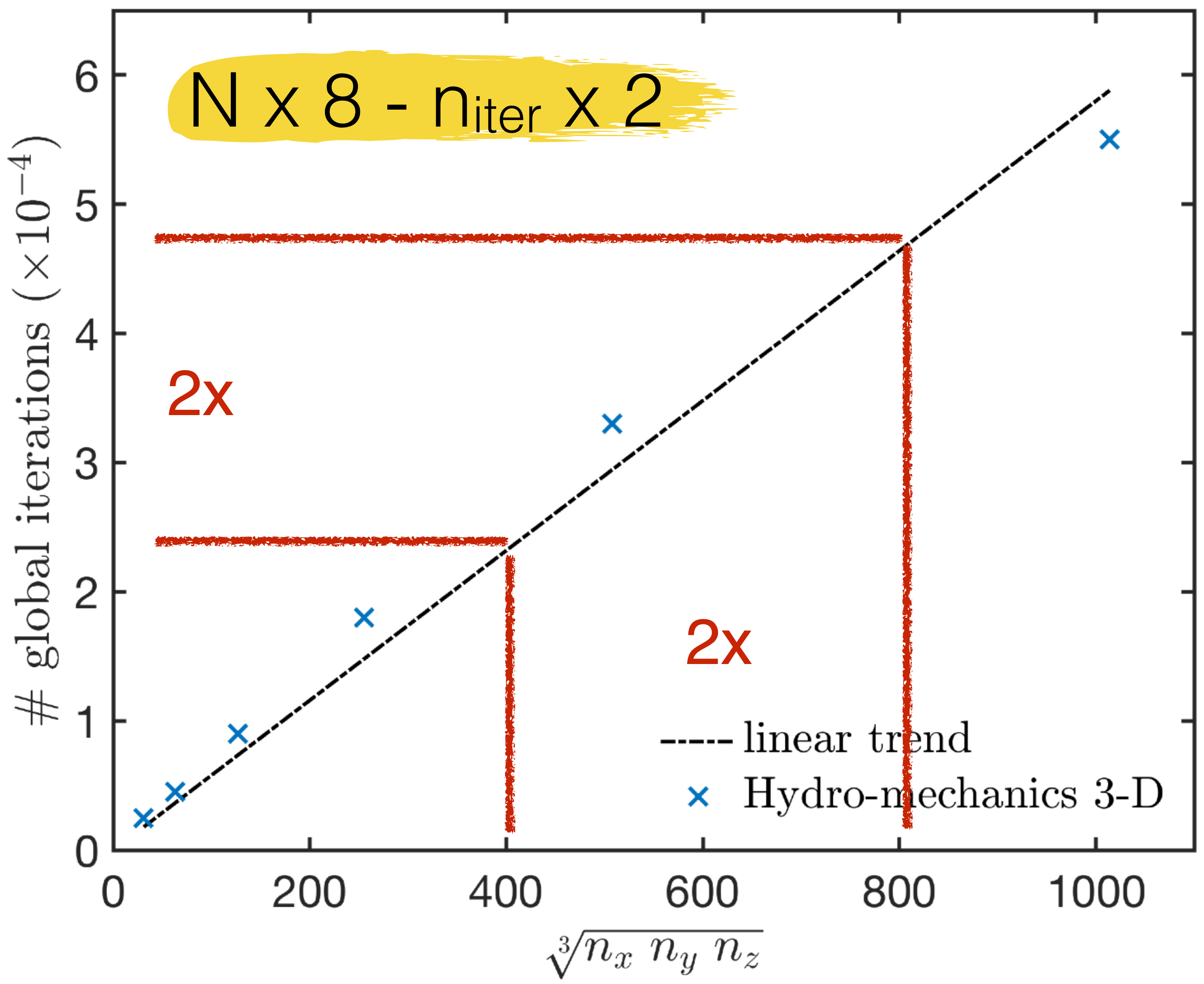
Iteration count ≠ time to solution

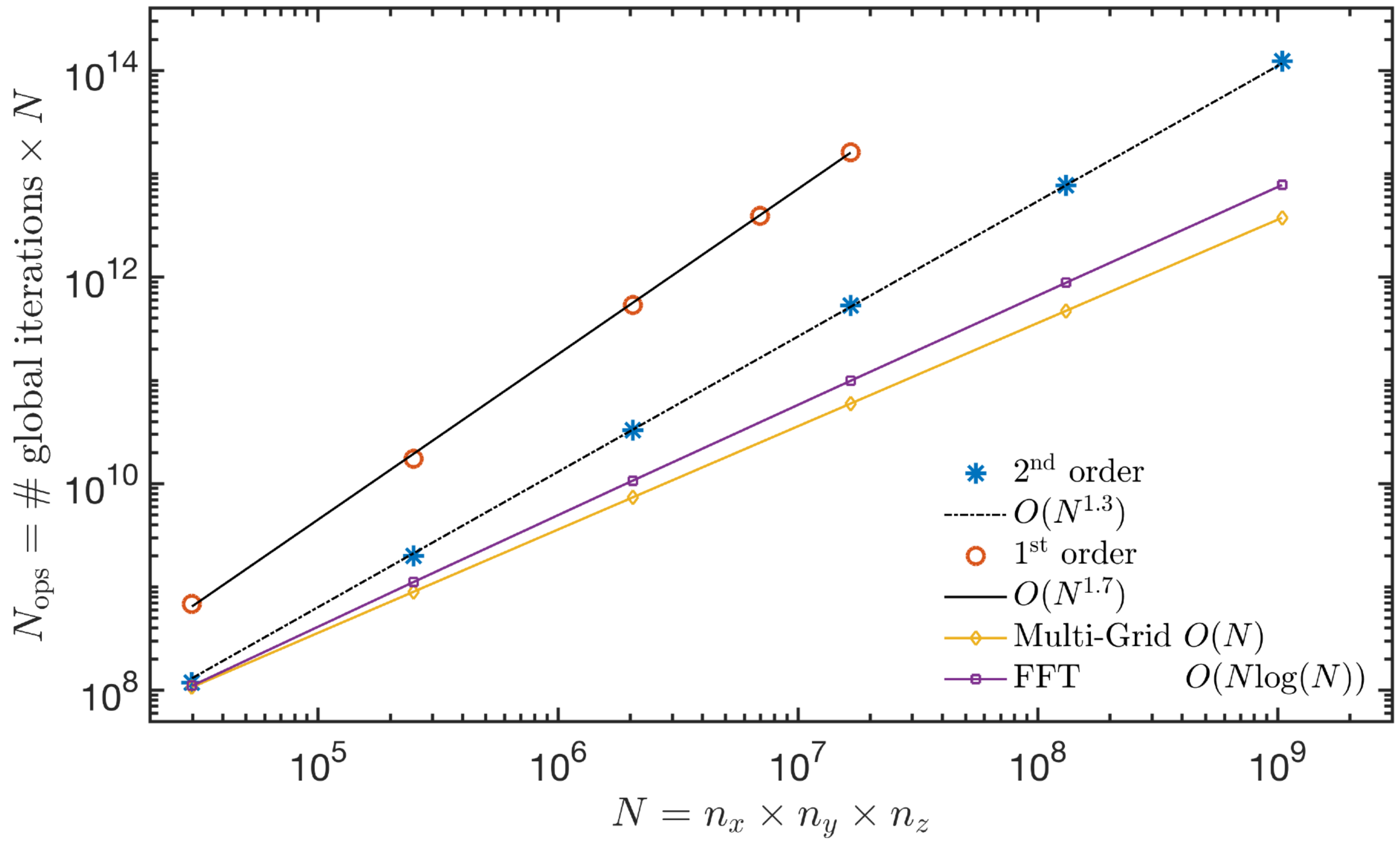
- Comparison to algorithms with ideal scalability



Iteration count ≠ time to solution

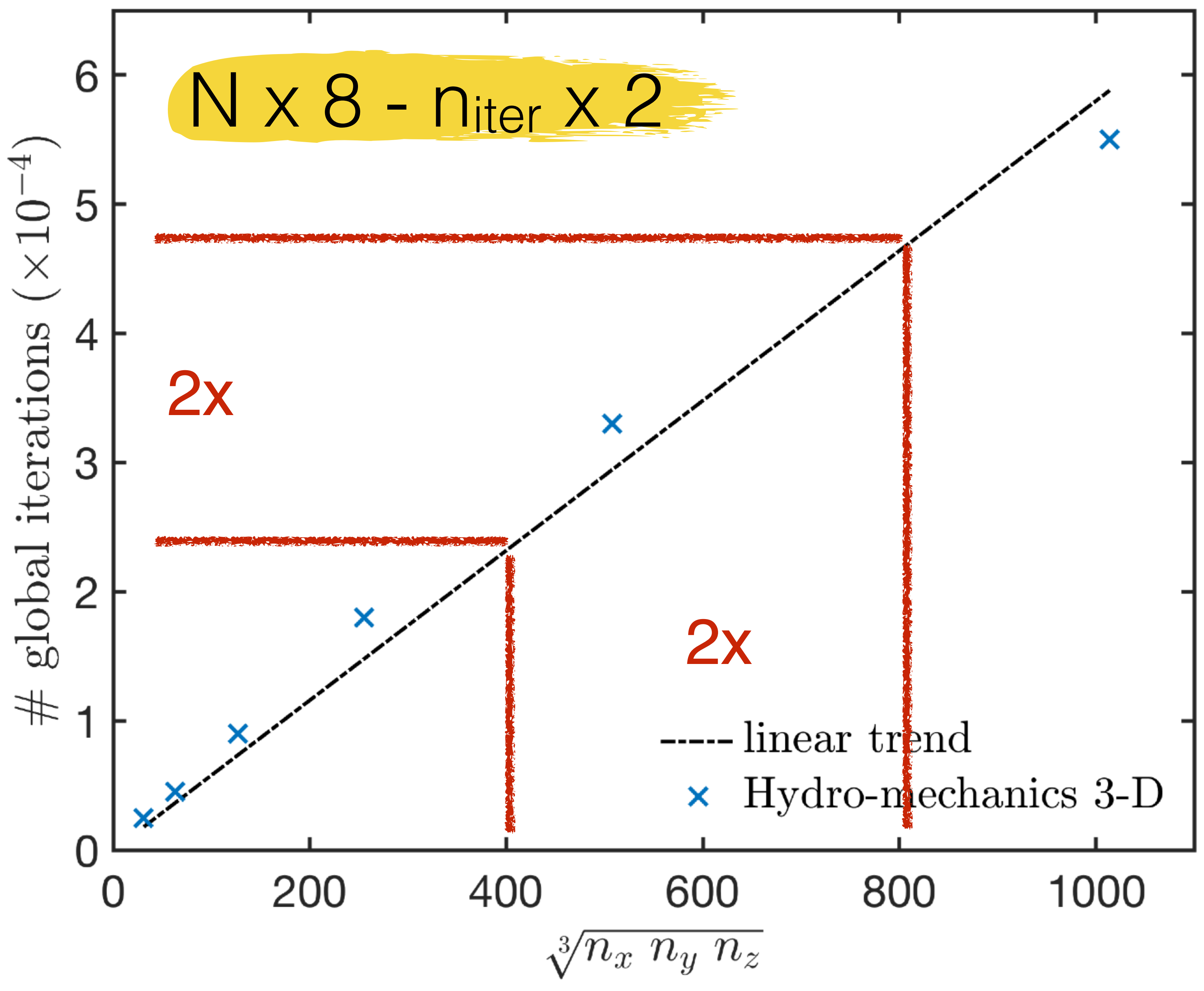
- Comparison to algorithms with ideal scalability

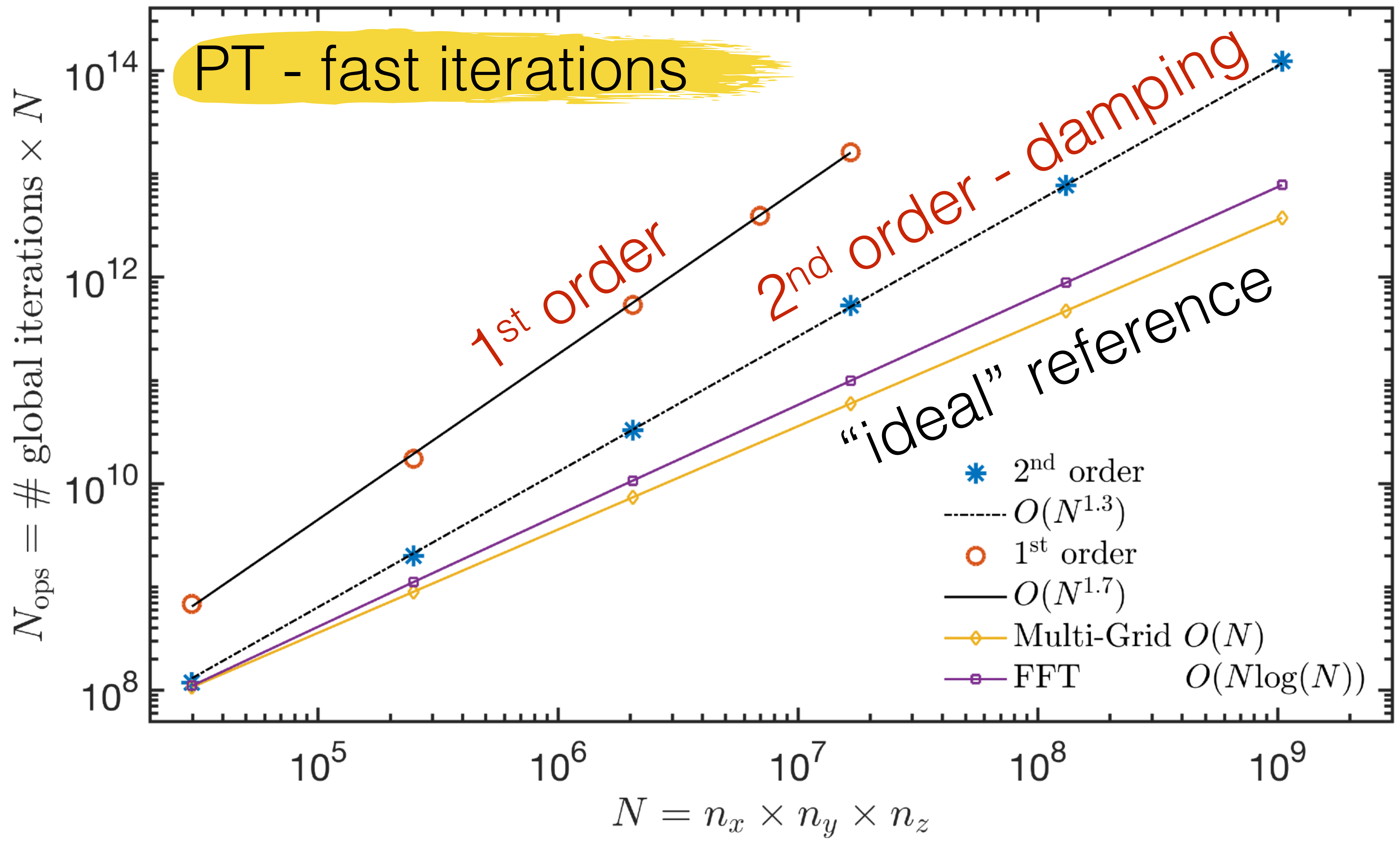




Iteration count ≠ time to solution

- Comparison to algorithms with ideal scalability





1/ Pseudo-Transient

Summary

- Pseudo-transient brings back transient physics and converges fast: $O(N^{1.3})$
- Second order - no additional communication (still local)
- But … additional memory transfers (past residual field [k-1] - damping)
- No need to gather information across the entire domain

1/ Pseudo-Transient

Summary

- Pseudo-transient brings back transient physics and converges fast: $O(N^{1.3})$
- Second order - no additional communication (still local)
- But … additional memory transfers (past residual field [k-1] - damping)
- No need to gather information across the entire domain

Physics is local & parallel - like nature

2/ Parallel efficiency

Distributed memory | large 3-D problem does not fit in a single GPU memory

- MPI: domain decomposition
- Send boundary points to neighbours as B.C.
- 6 planes to send in 3-D per sub-domain
- Volume to area ratio is the killer in 3-D
- Parallel efficiency will drop with more procs

  - network congestion, many HPC users, …

Conceptual 2-D domain decomposition

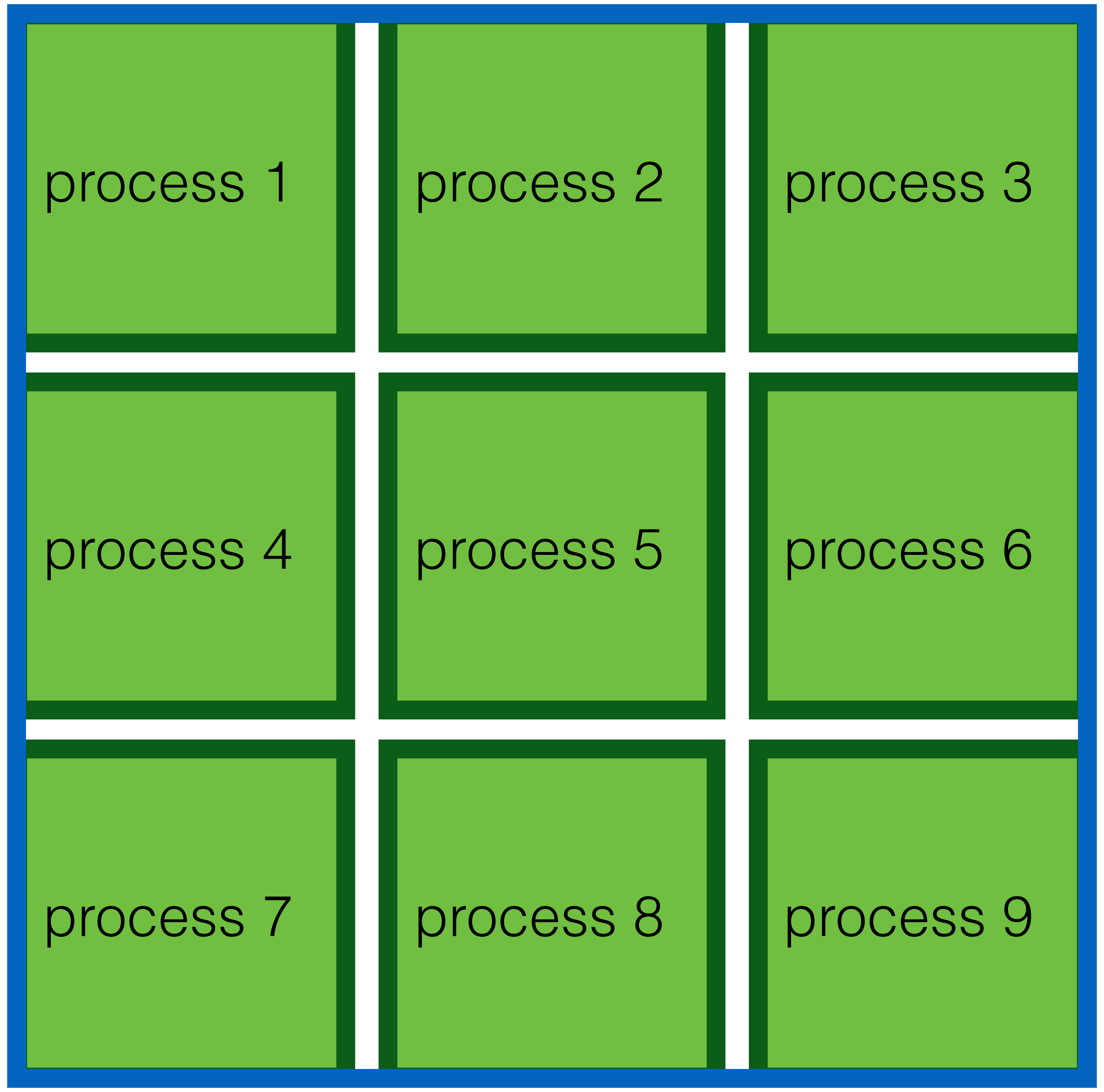
2/ Parallel efficiency

Distributed memory | large 3-D problem does not fit in a single GPU memory

- MPI: domain decomposition
- Send boundary points to neighbours as B.C.
- 6 planes to send in 3-D per sub-domain
- Volume to area ratio is the killer in 3-D
- Parallel efficiency will drop with more procs

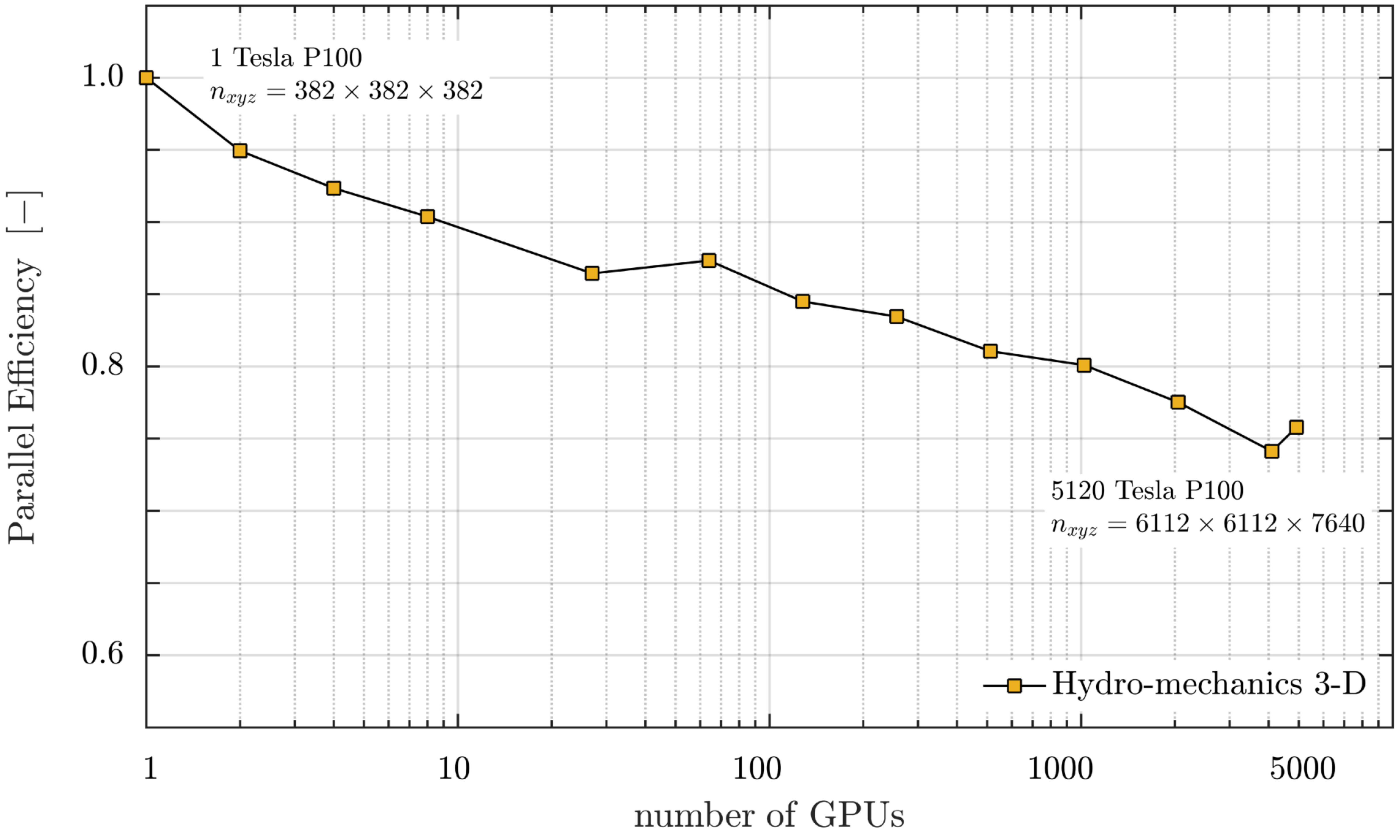
  - network congestion, many HPC users, …

Conceptual 2-D domain decomposition



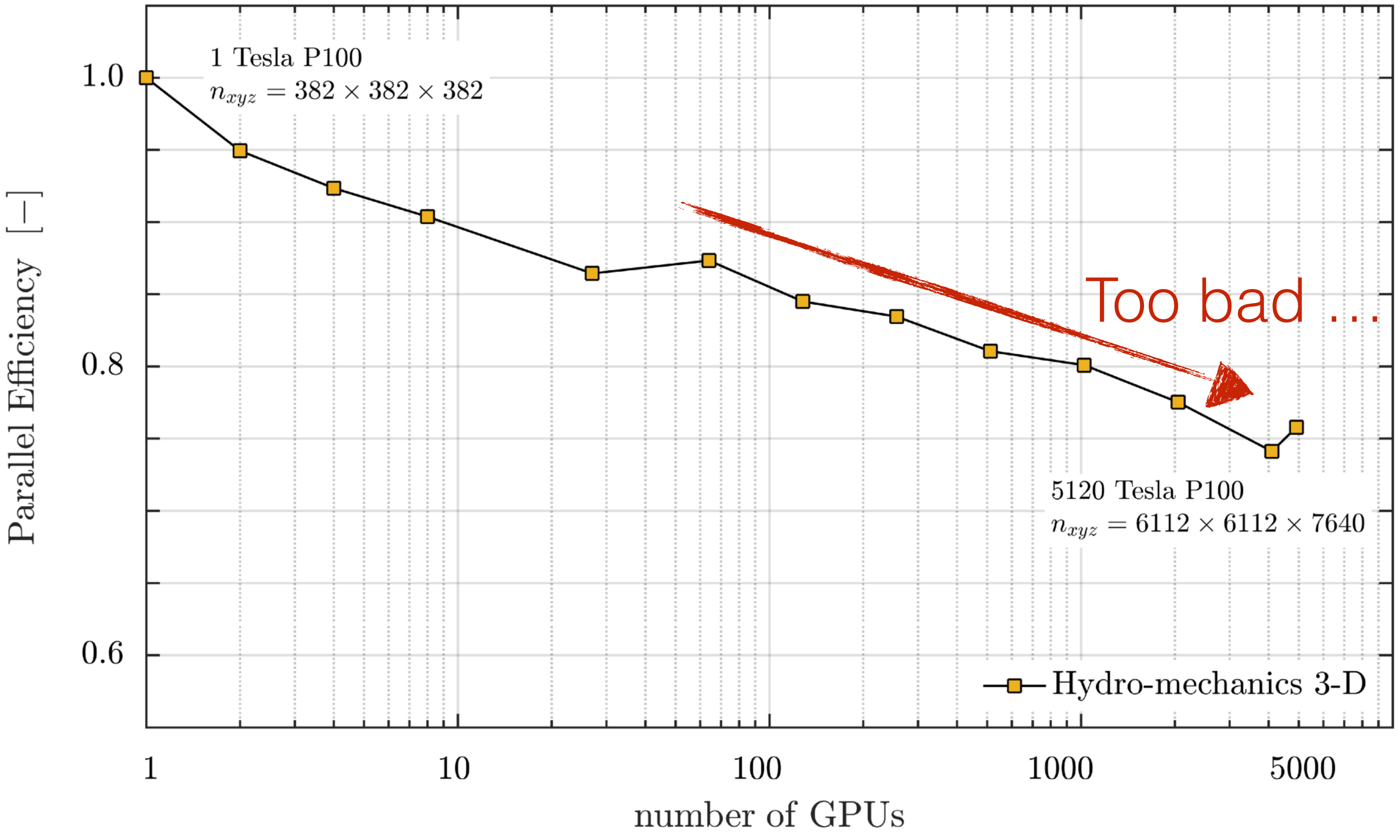
Costs of communication increase

- What will naively happen (weak scaling - physics efficiency = 1)



Costs of communication increase

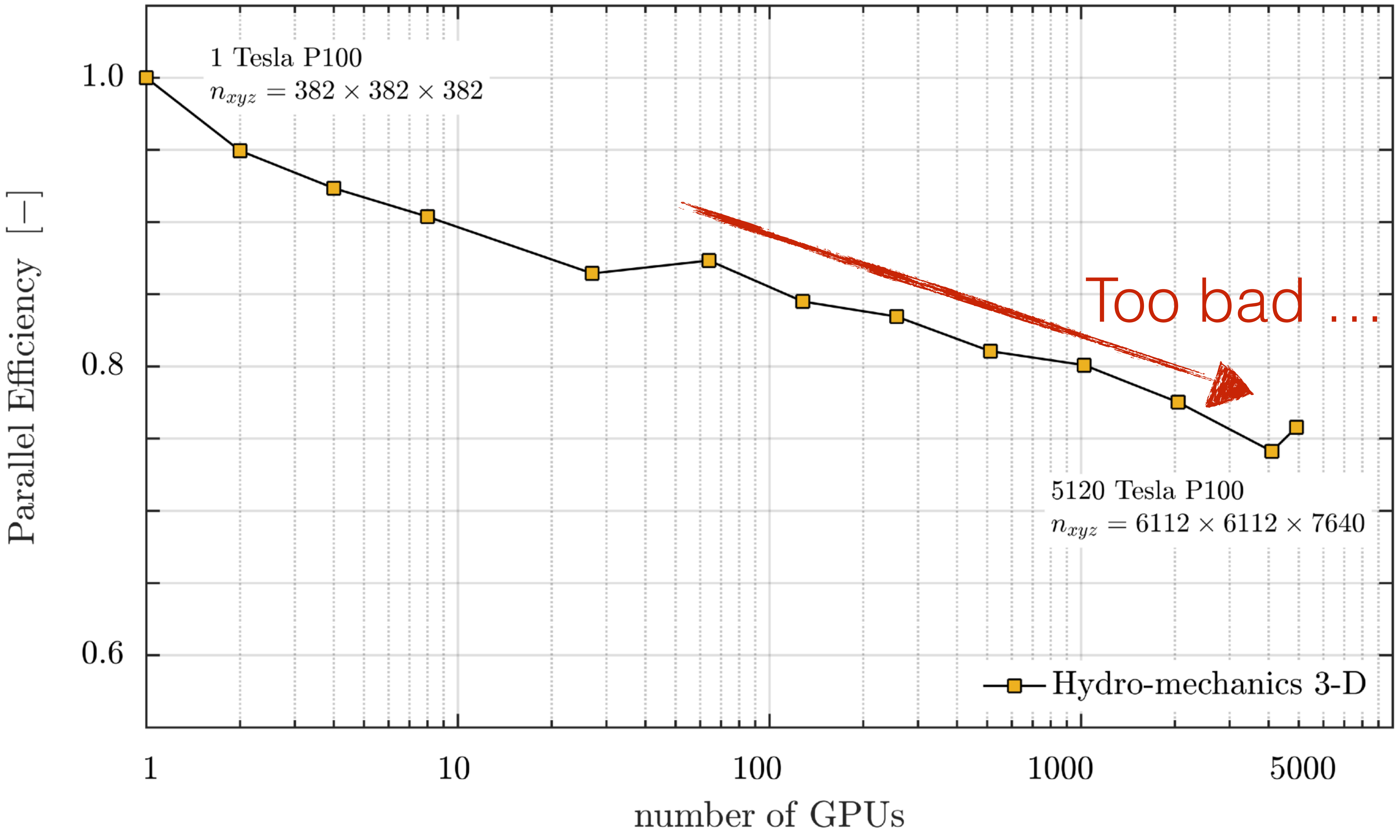
- What will naively happen (weak scaling - physics efficiency = 1)



Costs of communication increase

- What will naively happen (weak scaling - physics efficiency = 1)

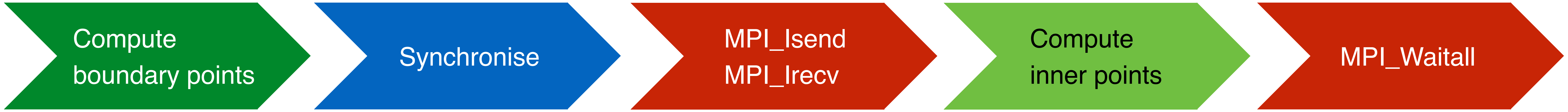
Solution: hide MPI communication

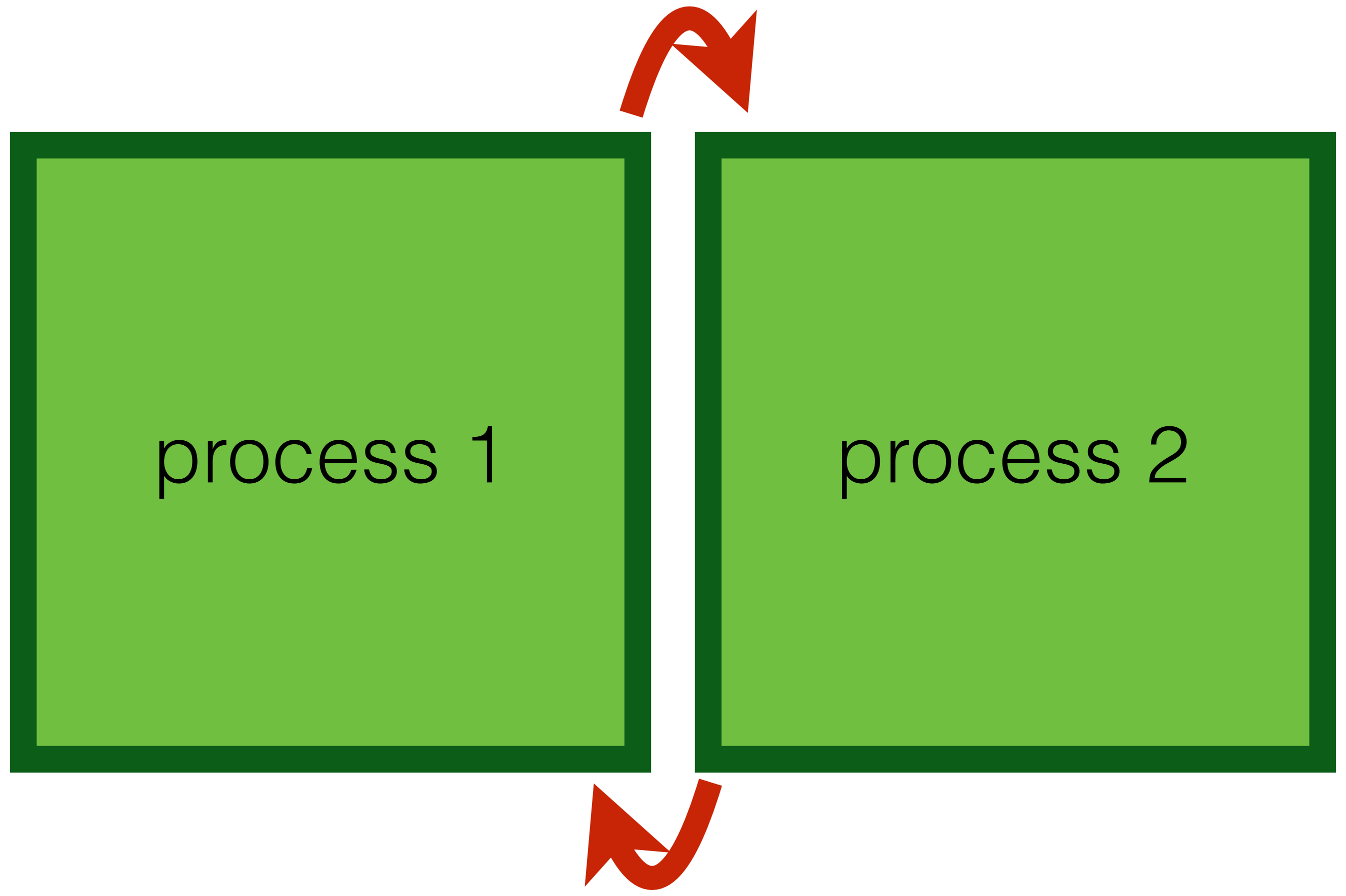


Hiding communication

- The simple concept:

Overlap boundary node communication with inner points computations

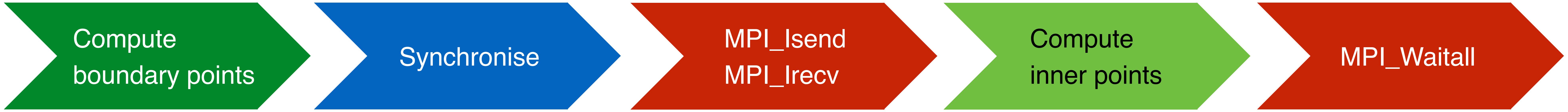




Hiding communication

- The simple concept:

Overlap boundary node communication with inner points computations



Conceptually simple, but no “easy solution” for real-world applications

Hiding communication

- The simple concept:

Overlap boundary node communication with inner points computations

MPI message progression in background will not continue
Need e.g. to call MPI_test to keep it alive … [1]

[1] *Hoefler and Lumsdaine, 2008. IEEE on cluster comp.*

Hiding communication

- The simple concept:

Overlap boundary node communication with inner points computations

MPI message progression in background will not continue
Need e.g. to call MPI_test to keep it alive … [1]

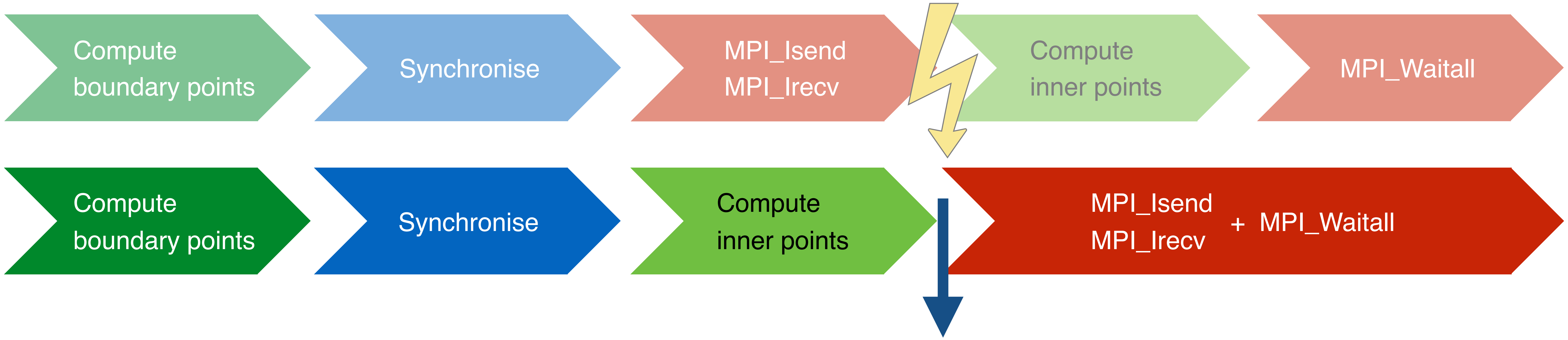
- The “MPI_test” approach may be complicated to do for a real world application...

[1] Hoefler and Lumsdaine, 2008. IEEE on cluster comp.

Hiding communication

- The simple concept:

Overlap boundary node communication with inner points computations

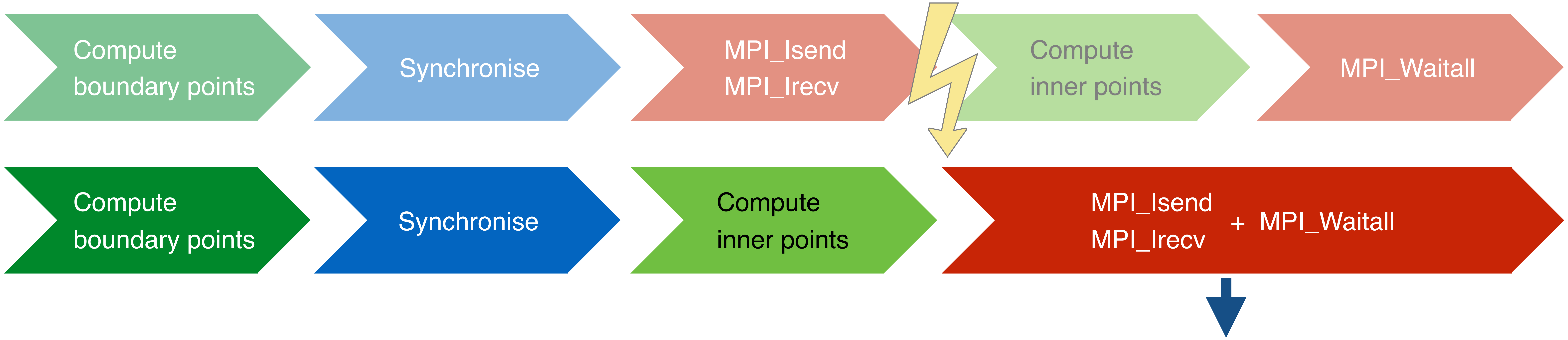


Control goes back to host thread, next step begins directly.
The host threads blocks only when reaching MPI_Waitall:
> trivially ensures persistent message progression (basic MPI)

Hiding communication

- The simple concept:

Overlap boundary node communication with inner points computations



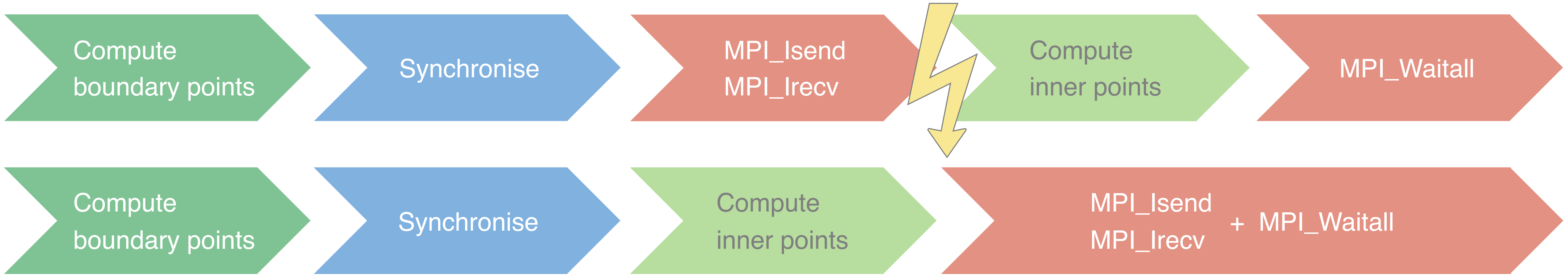
Synchronise + call MPI_Waitall freely in this function

/!\ Only avoid blocking the GPU kernel [Compute inner points] !

Hiding communication

- The simple concept:

Overlap boundary node communication with inner points computations

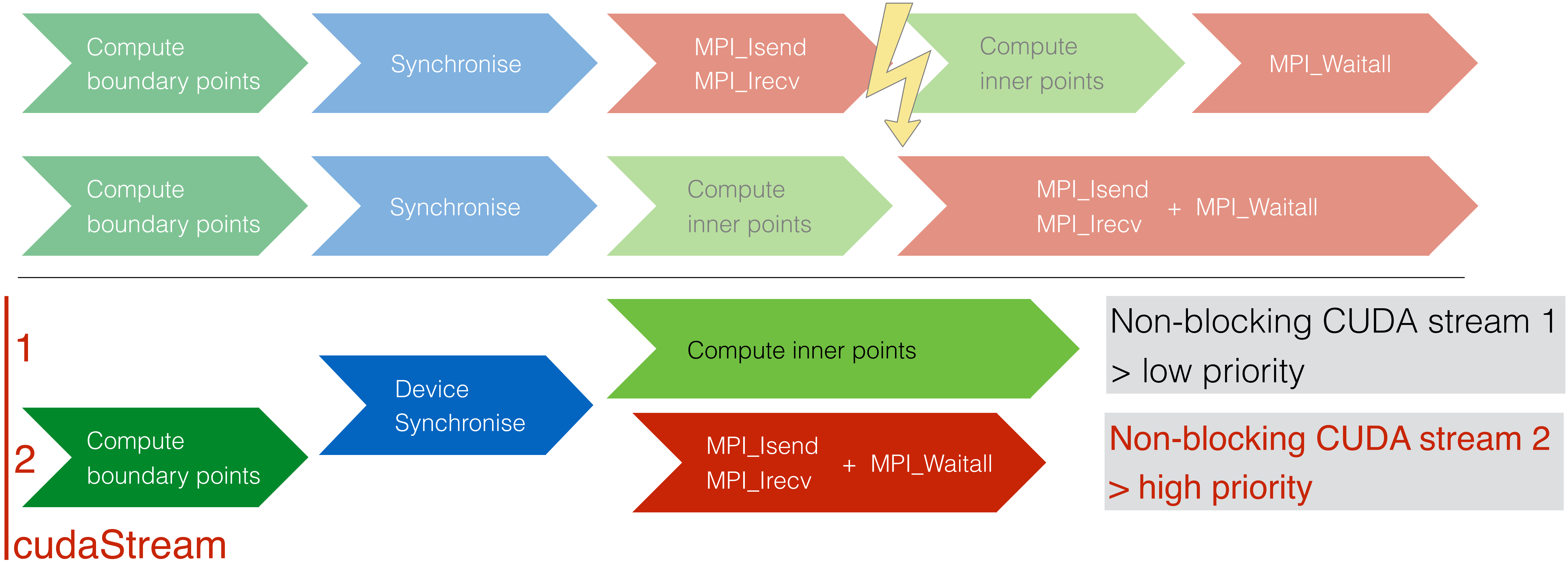


- How to access boundary data on the GPU without blocking the ongoing computations ?
- How to make this fast in order to quickly initiate the MPI data transfers ?

Hiding communication

- The simple concept:

Overlap boundary node communication with inner points computations

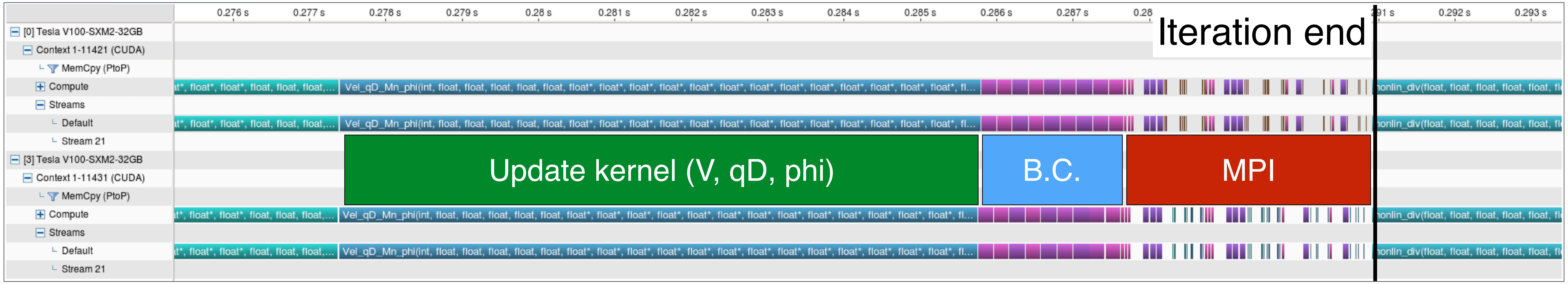


Visual profiler (nvvp)

Original

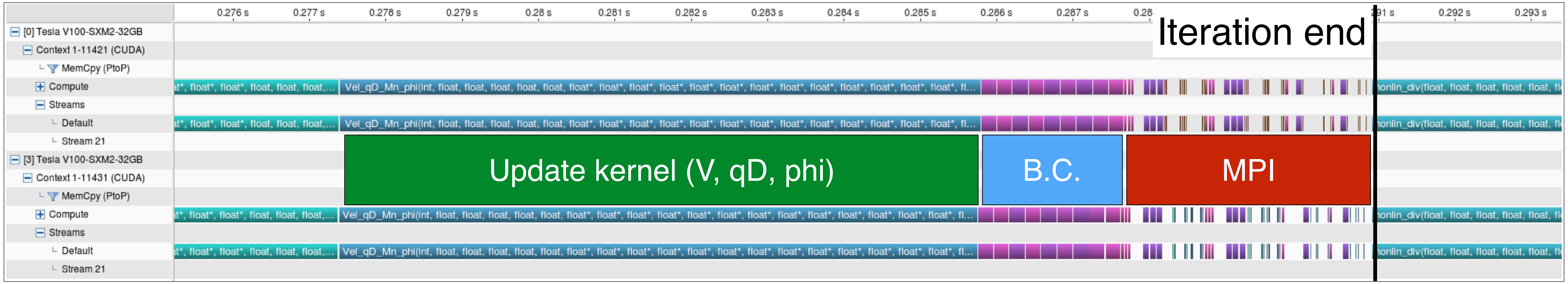
Visual profiler (nvvp)

Original

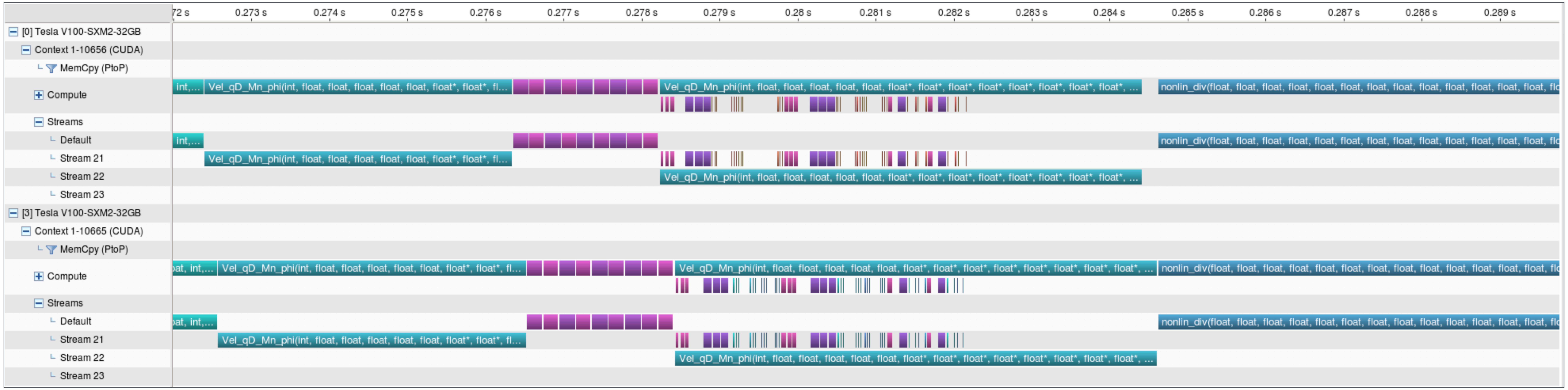


Visual profiler (nvvp)

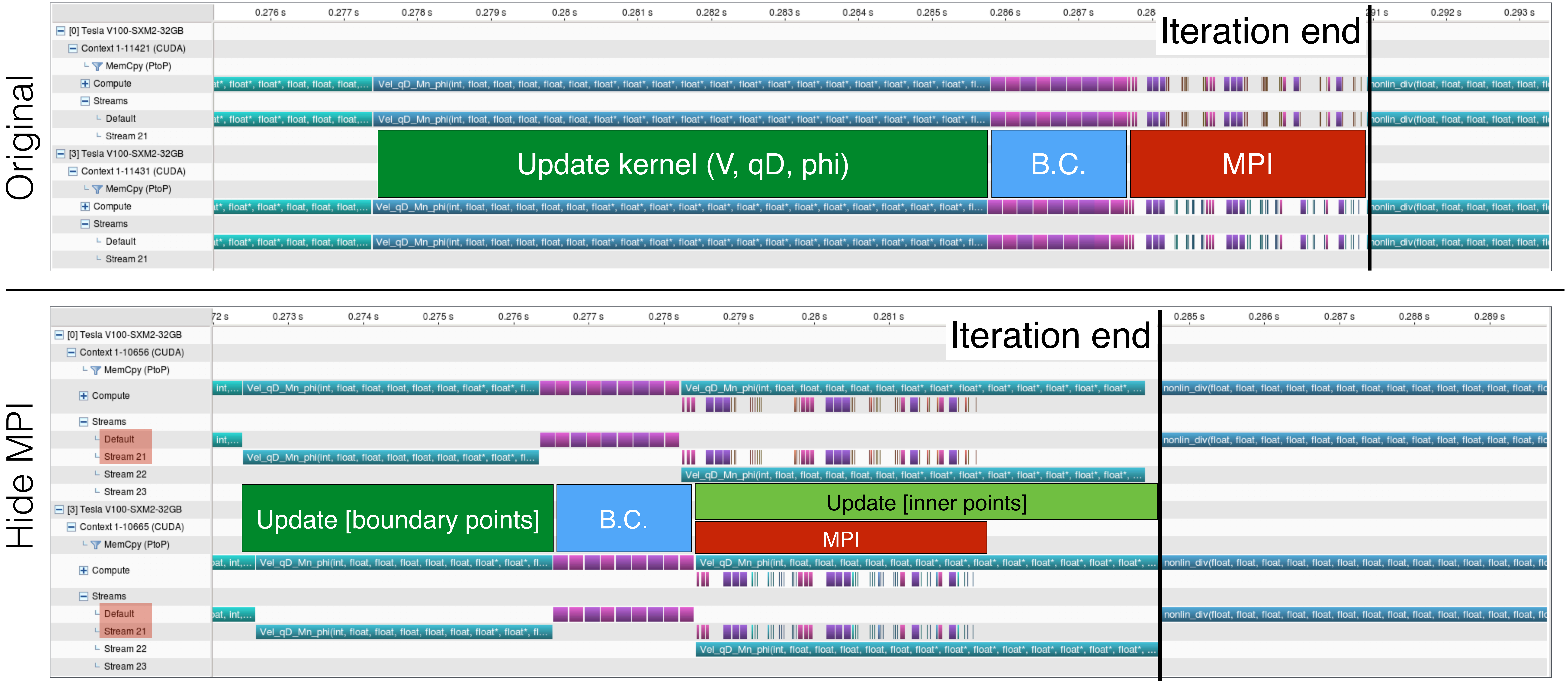
Original



Hide MPI

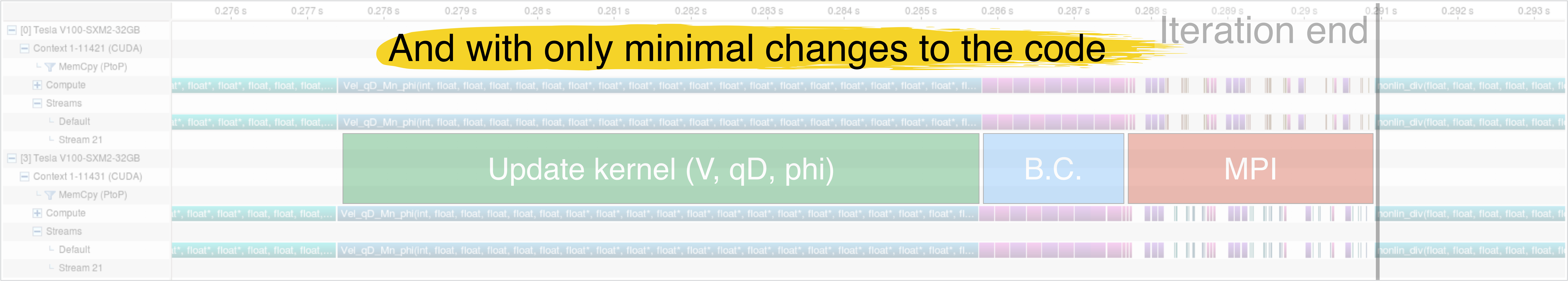


Visual profiler (nvvp)

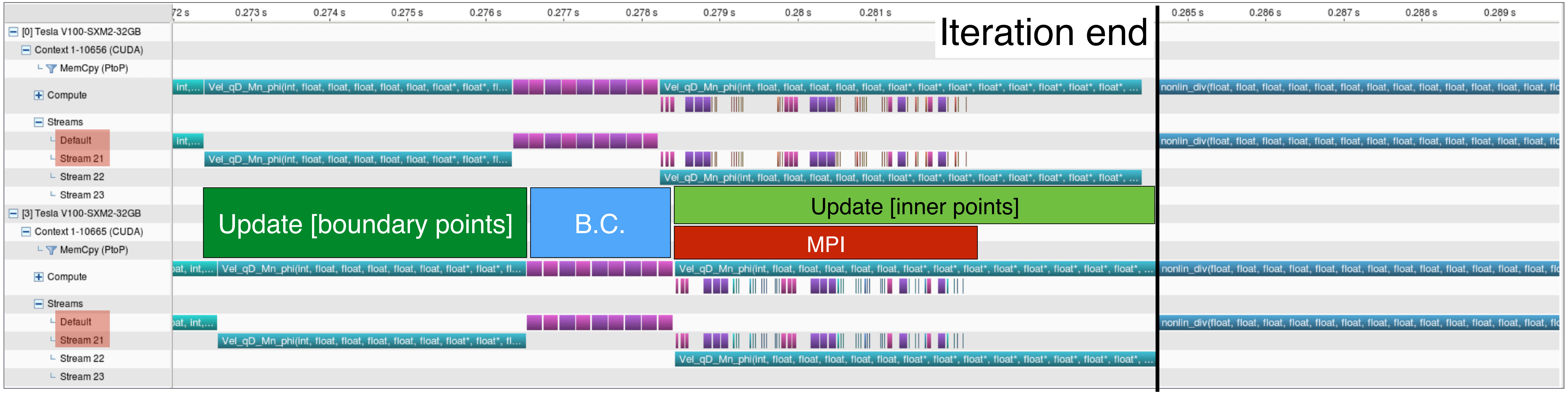


Visual profiler (nvvp)

Original

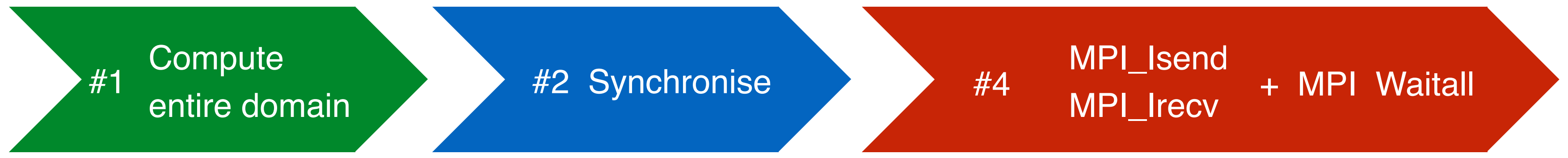


Hide MPI



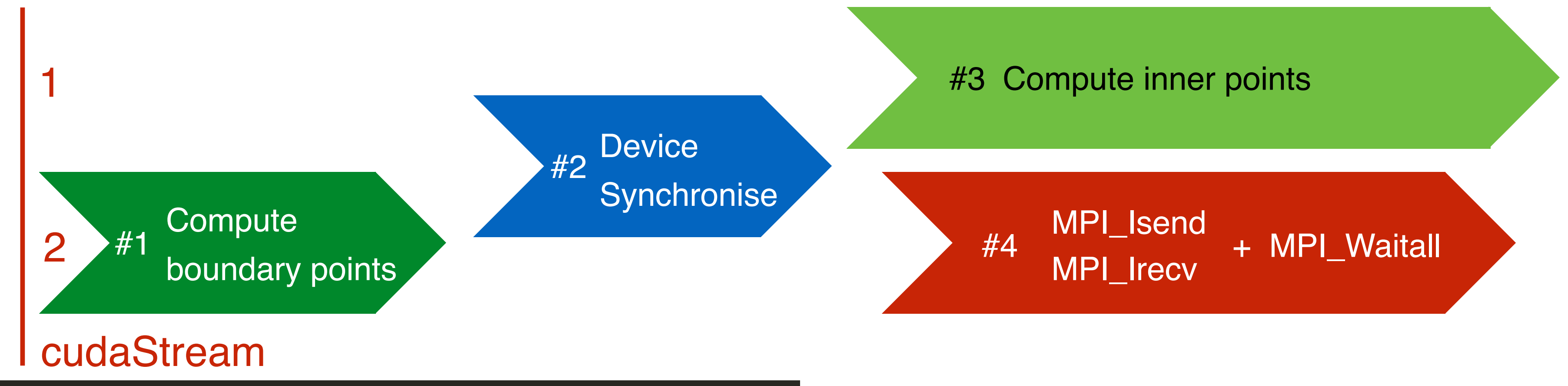
Minimal changes to the code

```
__global__ void compute(int istep){
    int ix = blockIdx.x*blockDim.x + threadIdx.x; // thread ID, dimension x
    int iy = blockIdx.y*blockDim.y + threadIdx.y; // thread ID, dimension y
    int iz = blockIdx.z*blockDim.z + threadIdx.z; // thread ID, dimension z

    //(...) (rest of the kernel as before)
}

main(){
    //(...)

        compute<<<grid,block                 >>>(      );     // step 1
                    cudaDeviceSynchronize();                   // step 2

    update_boundaries();                                       // step 4
    //(...)
}
```



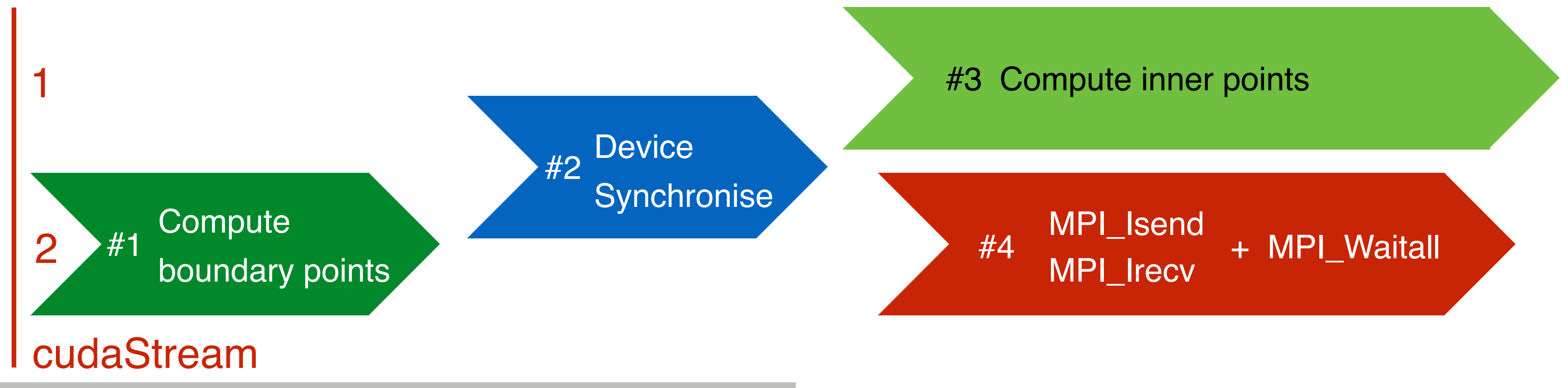
Minimal changes to the code

```
#define BOUNDARY_WIDTH_X 32
#define BOUNDARY_WIDTH_Y 32
#define BOUNDARY_WIDTH_Z 32

__global__ void compute(int istep){
    int ix = blockIdx.x*blockDim.x + threadIdx.x; // thread ID, dimension x
    int iy = blockIdx.y*blockDim.y + threadIdx.y; // thread ID, dimension y
    int iz = blockIdx.z*blockDim.z + threadIdx.z; // thread ID, dimension z
    // Return if no work to do.
    if (  istep==0 &&     ix>=BOUNDARY_WIDTH_X && ix<=nx-1-BOUNDARY_WIDTH_X &&
                          iy>=BOUNDARY_WIDTH_Y && iy<=ny-1-BOUNDARY_WIDTH_Y &&
                          iz>=BOUNDARY_WIDTH_Z && iz<=nz-1-BOUNDARY_WIDTH_Z    ) return;
    if (  istep==1 && (   ix< BOUNDARY_WIDTH_X || ix> nx-1-BOUNDARY_WIDTH_X ||
                          iy< BOUNDARY_WIDTH_Y || iy> ny-1-BOUNDARY_WIDTH_Y ||
                          iz< BOUNDARY_WIDTH_Z || iz> nz-1-BOUNDARY_WIDTH_Z  ) ) return;
    //(...) (rest of the kernel as before)
}

main(){
    //(...)
    for (int istep=0; istep<2; istep++){
        compute<<<grid,block,0,streams[istep]>>>(istep);      // step 1 (istep==0) and step 3 (istep==1)
        if (istep==0) cudaDeviceSynchronize();                // step 2
    }
    update_boundaries();                                      // step 4
    //(...)
}
```



Minimal changes to the code

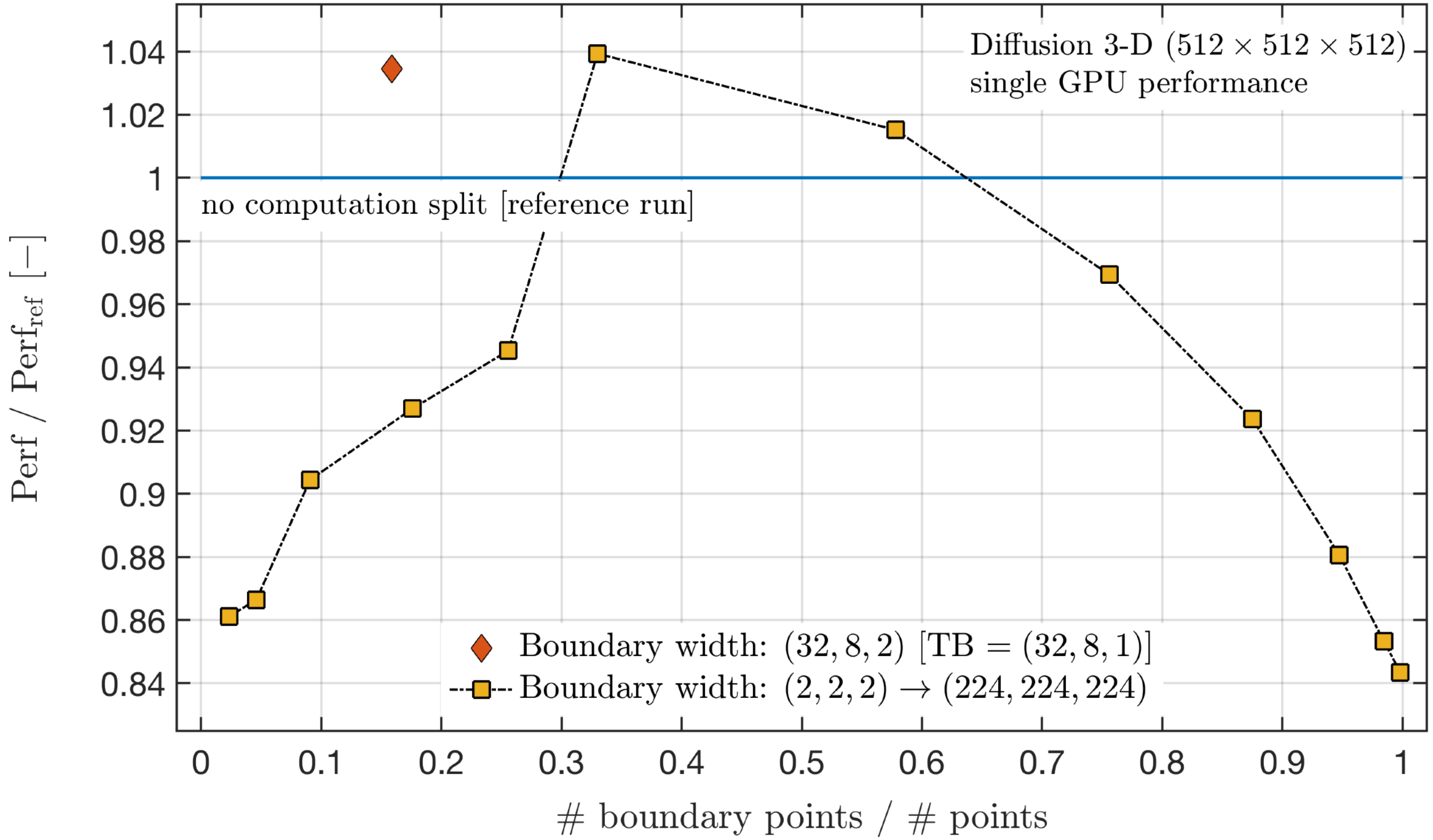
```
#define BOUNDARY_WIDTH_X 32
#define BOUNDARY_WIDTH_Y 32
#define BOUNDARY_WIDTH_Z 32

__global__ void compute(int istep){
    int ix = blockIdx.x*blockDim.x + threadIdx.x; // thread ID, dimension x
    int iy = blockIdx.y*blockDim.y + threadIdx.y; // thread ID, dimension y
    int iz = blockIdx.z*blockDim.z + threadIdx.z; // thread ID, dimension z
    // Return if no work to do.
    if (  istep==0 &&     ix>=BOUNDARY_WIDTH_X && ix<=nx-1-BOUNDARY_WIDTH_X &&
                          iy>=BOUNDARY_WIDTH_Y && iy<=ny-1-BOUNDARY_WIDTH_Y &&
                          iz>=BOUNDARY_WIDTH_Z && iz<=nz-1-BOUNDARY_WIDTH_Z    ) return;
    if (  istep==1 && (   ix< BOUNDARY_WIDTH_X || ix> nx-1-BOUNDARY_WIDTH_X ||
                          iy< BOUNDARY_WIDTH_Y || iy> ny-1-BOUNDARY_WIDTH_Y ||
                          iz< BOUNDARY_WIDTH_Z || iz> nz-1-BOUNDARY_WIDTH_Z  ) ) return;
    //(...) (rest of the kernel as before)
}

main(){
    //(...)
    for (int istep=0; istep<2; istep++){
        compute<<<grid,block,0,streams[istep]>>>(istep);     // step 1 (istep==0) and step 3 (istep==1)
        if (istep==0) cudaDeviceSynchronize();              // step 2
    }
    update_boundaries();                                    // step 4
    //(...)
}
```



Avoid loss of performance when splitting boundary: Make boundary width > 2

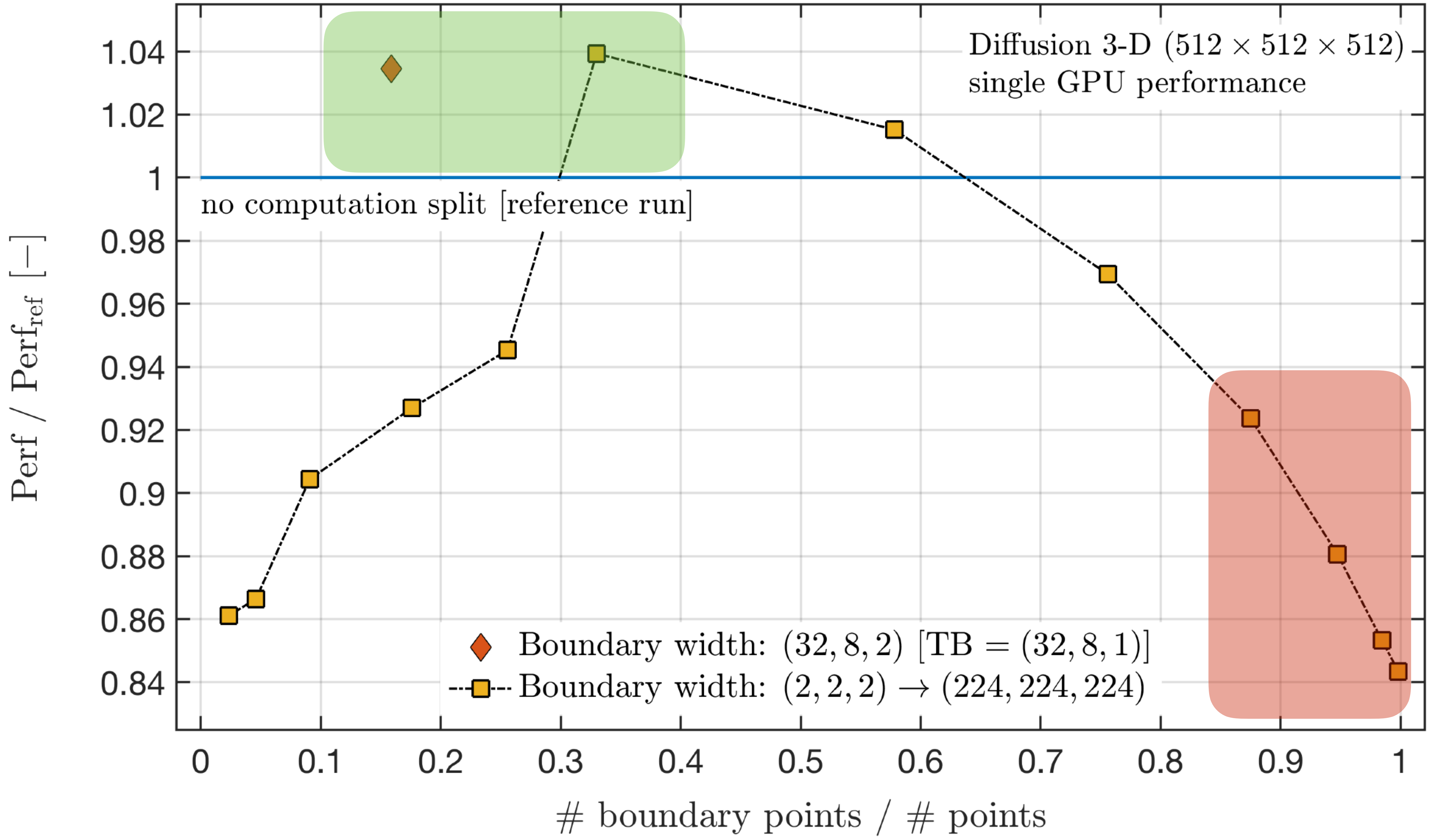
Avoid loss of performance

- Impact of boundary width on single GPU performance



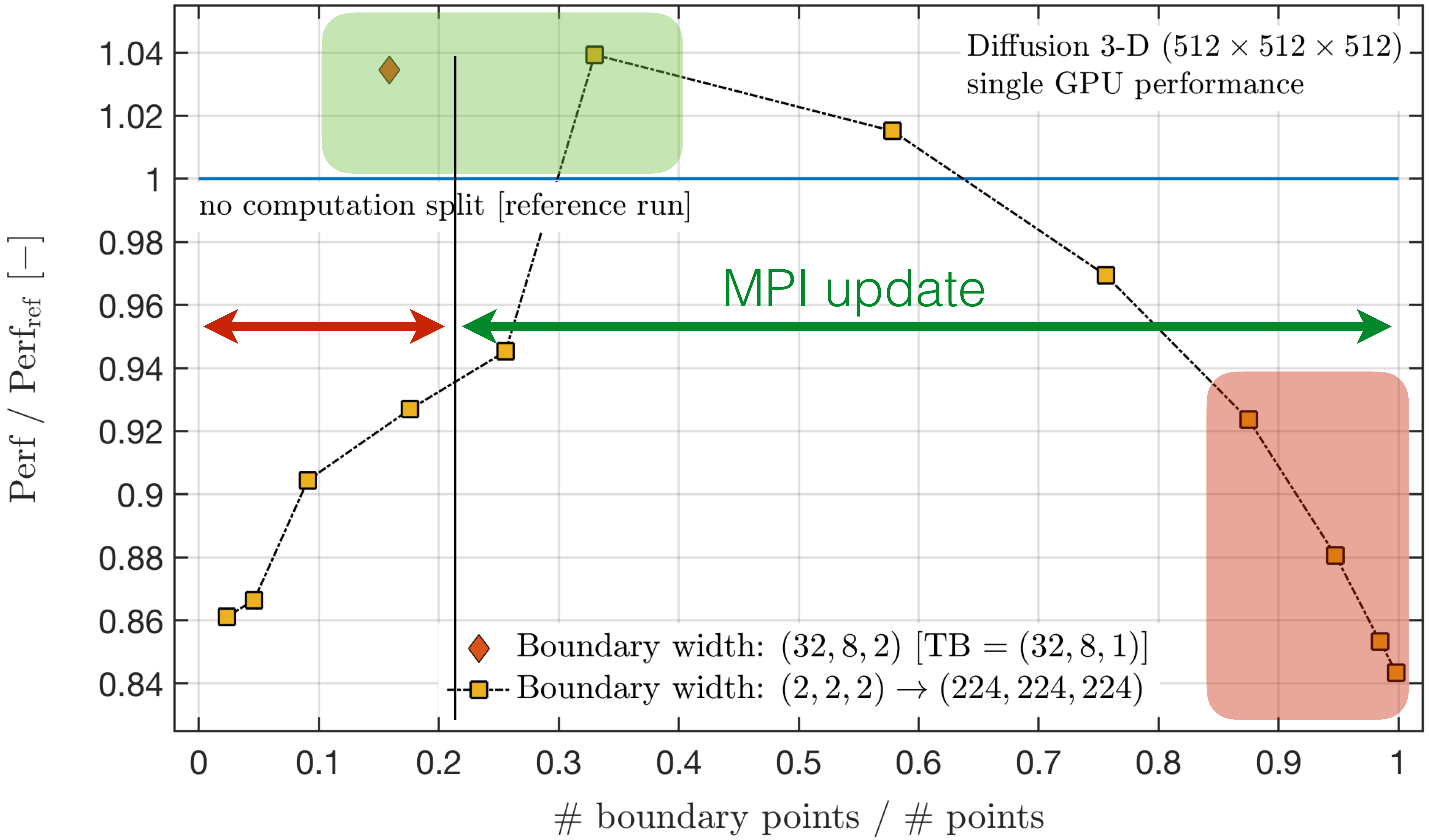
Avoid loss of performance

- Impact of boundary width on single GPU performance



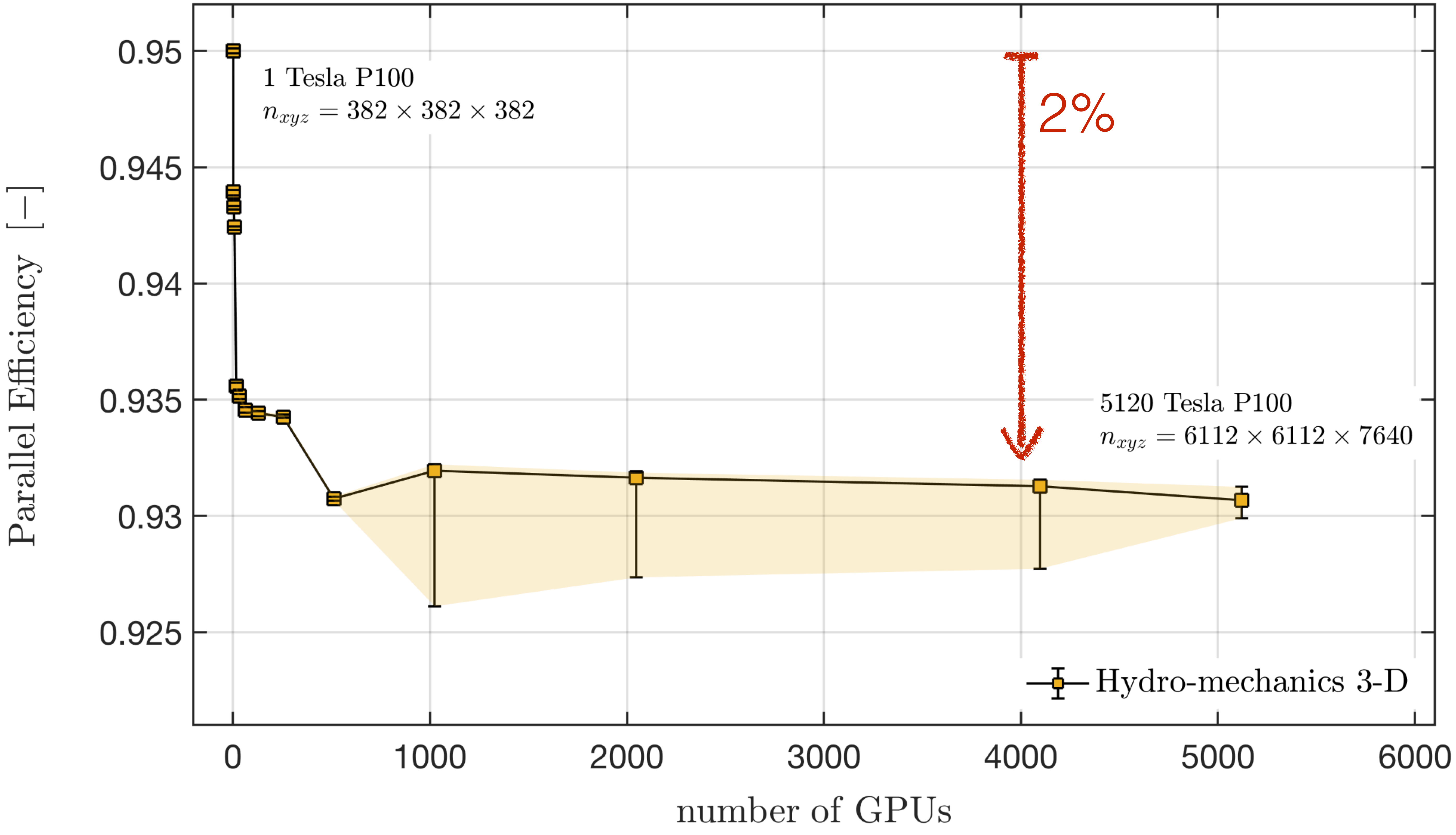
Avoid loss of performance

- Impact of boundary width on single GPU performance



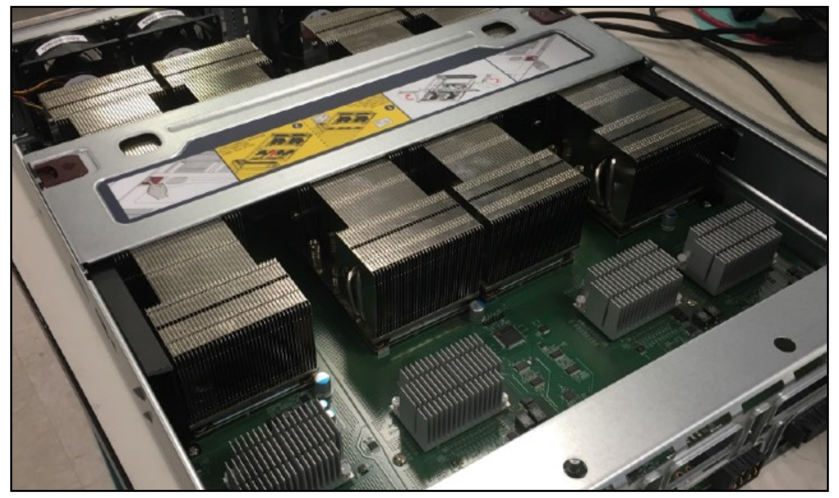
Parallel efficiency

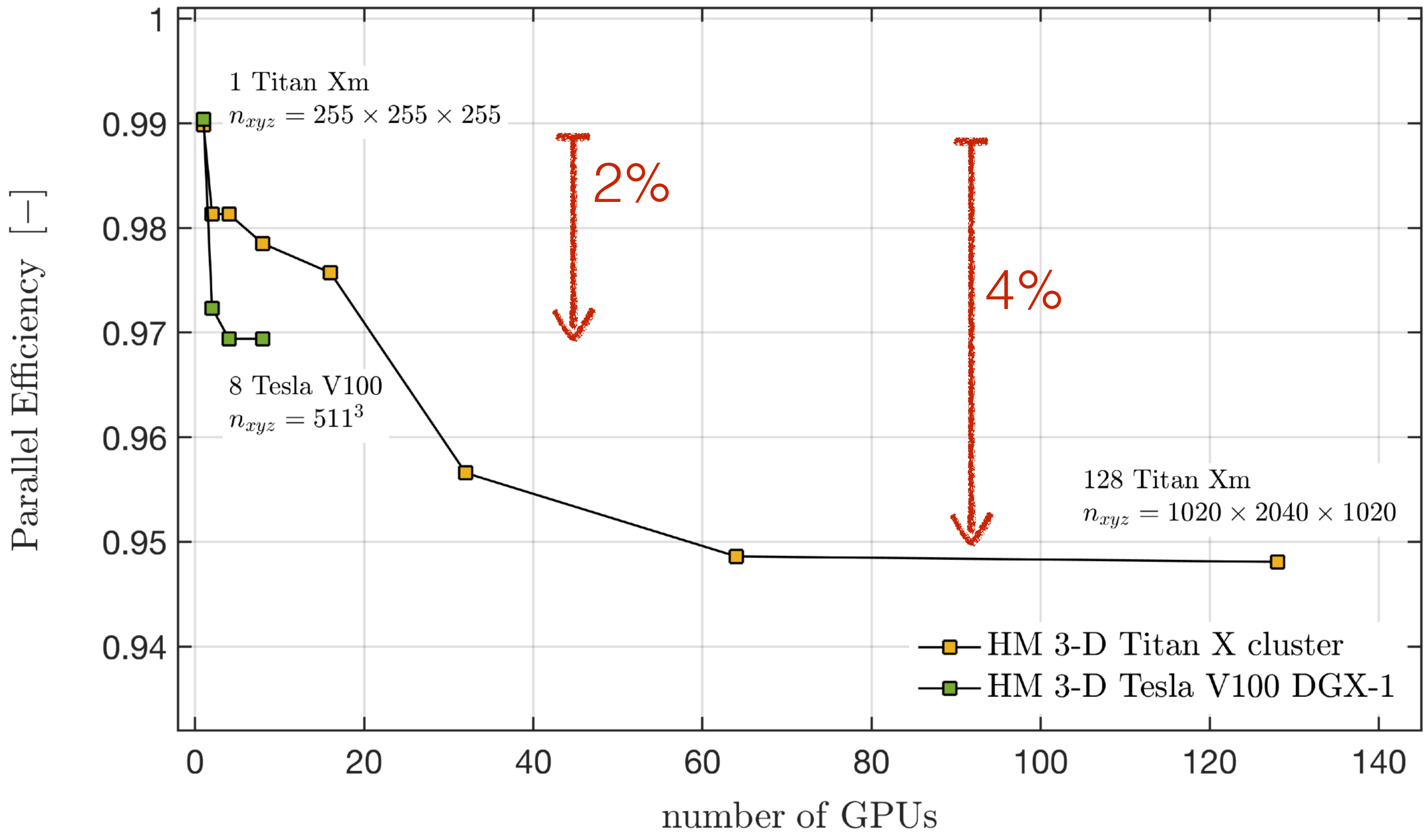
- 2% drop on 5120 Tesla P100 GPUs on Cray XC50 "Piz Daint" @ CSCS



Parallel efficiency

- In-house: 4% drop on 128 Titan Xm GPUs on in-house supercomputer “Octopus” @ Unil
  2% drop on 8 Tesla V100 Nvlink GPUs (DGX-1) “Volta” @ Unil





2/ Parallel efficiency

Summary

- Only 2% drop of parallel efficiency on 5120 GPUs compared to single GPU execution
- CudaStream: simple approach to access GPU memory while computations are ongoing
- Boundary width > 2, can avoid loss of performance when splitting computations
- Avoid blocking the computing and prioritise boundary update workflow
- Only minimal changes to original code - even for real-world applications !

Summary

We tackled hardware limit with our multi-GPU 3-D solver

- 1/ Using simple physics-based fast local iterative scheme
  > Pseudo-time brings physics back

- 2/ Implementing domain decomposition and asynchronous execution to hide MPI
  > Scale on the world's fastest supercomputers

Summary

We tackled hardware limit with our multi-GPU 3-D solver

- 1/ Using simple physics-based fast local iterative scheme
  > Pseudo-time brings physics back

- 2/ Implementing domain decomposition and asynchronous execution to hide MPI
  > Scale on the world's fastest supercomputers

A general approach that is not restricted to our particular geo-physics application

Outlook

This approach can be applied to a wide range of computational physics problems

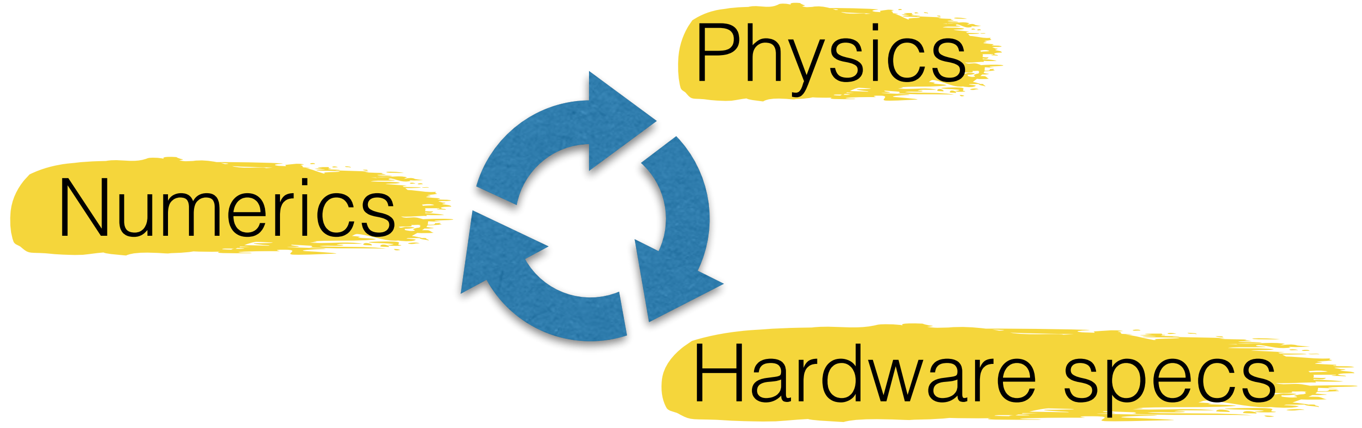
- Get the most out of a tight interplay

- Improve the effective memory throughput of our solver | MTP_{eff} @ ~25% of memcpy
- What's next: Testing PT iterations for for Finite-Elements
- What's next: Application to various coupled physical processes in Earth sciences

Outlook

This approach can be applied to a wide range of computational physics problems

- Get the most out of a tight interplay



- Improve the effective memory throughput of our solver | MTP_{eff} @ ~25% of memcpy
- What’s next: Testing PT iterations for for Finite-Elements
- What’s next: Application to various coupled physical processes in Earth sciences

Thank you

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

COLFAX
Customized Solutions

Unil
UNIL | Université de Lausanne
Swiss Geocomputing Centre

FNSNF
FONDS NATIONAL SUISSE
SCHWEIZERISCHER NATIONALFONDS
FONDO NAZIONALE SVIZZERO
SWISS NATIONAL SCIENCE FOUNDATION

SIGMA

Stanford University

Thank you

wp.unil.ch/geocomputing/

lraess@stanford.edu

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

COLFAX
Customized Solutions

UNIL | Université de Lausanne
Swiss Geocomputing Centre

FNSNF
FONDS NATIONAL SUISSE
SCHWEIZERISCHER NATIONALFONDS
FONDO NAZIONALE SVIZZERO
SWISS NATIONAL SCIENCE FOUNDATION

SIGMA

References

Räss, L., Simon, N. S. C., & Podladchikov, Y. Y. (2018). Spontaneous formation of fluid escape pipes from subsurface reservoirs. *Scientific Reports*, *8*(1), 11116.

Løseth, H., Wensaas, L., Arntsen, B., Hanken, N.-M., Basire, C., & Graue, K. (2011). 1000 m long gas blow-out pipes. *Marine and Petroleum Geology*, *28*(5), 1047–1060.

Plaza-Faverola, A., Bünz, S., & Mienert, J. (2011). Repeated fluid expulsion through sub-seabed chimneys offshore Norway in response to glacial cycles. *Earth and Planetary Science Letters*, *305*(3–4), 297–308.

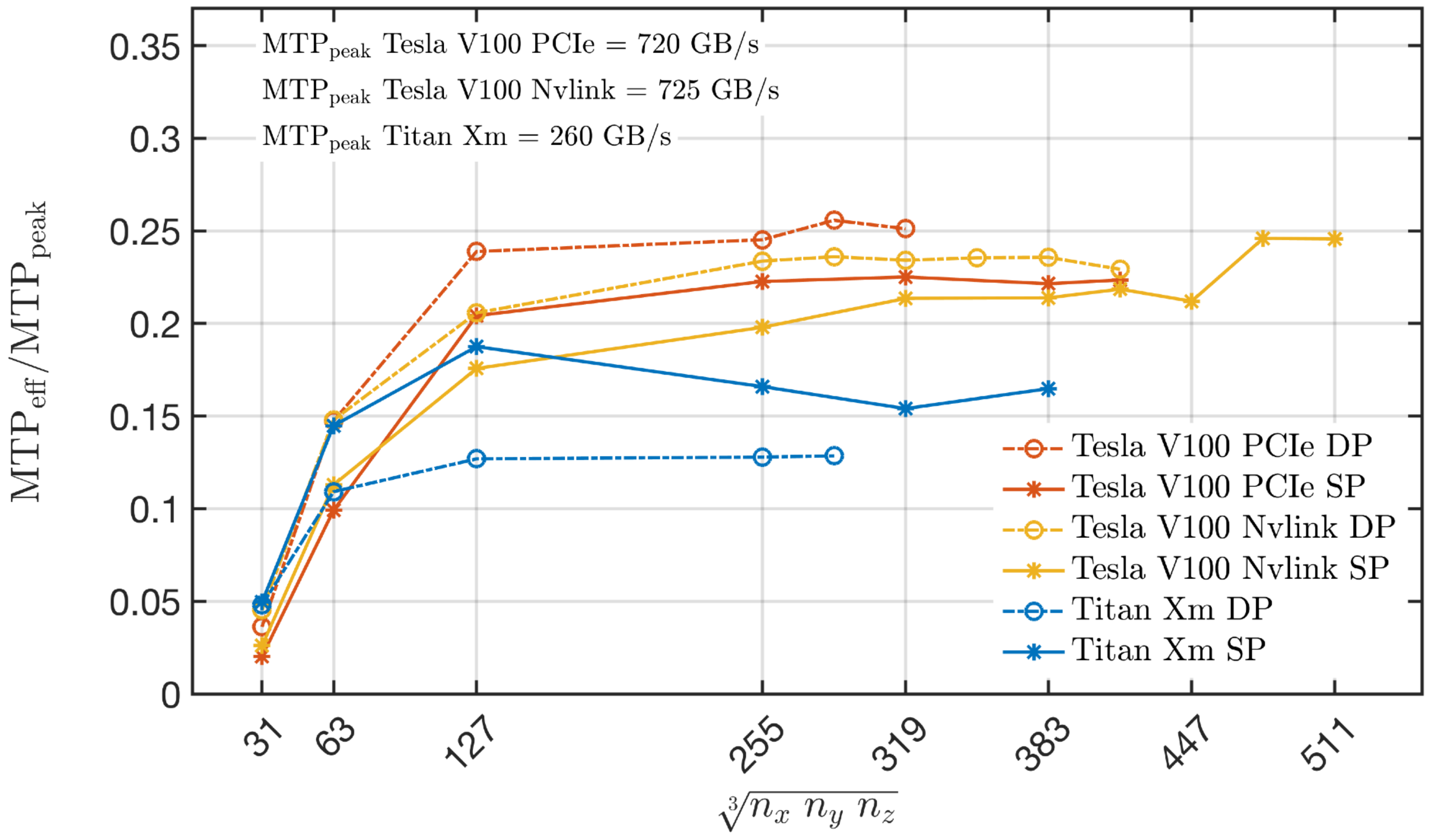
Hoefler, T., & Lumsdaine, A. (2008). Message progression in parallel computing - to thread or not to thread? In *2008 IEEE International Conference on Cluster Computing* (Vol. Proceeding, pp. 213–222). IEEE.

Frankel, S. P. (1950). Convergence rates of iterative treatments equations of partial differential. Mathematical Tables and Other Aids to Computation, 4(30), 65–75.

Effective memory throughput

Effective and absolute metric to measure optimality of data access: $MTP_{effective}$

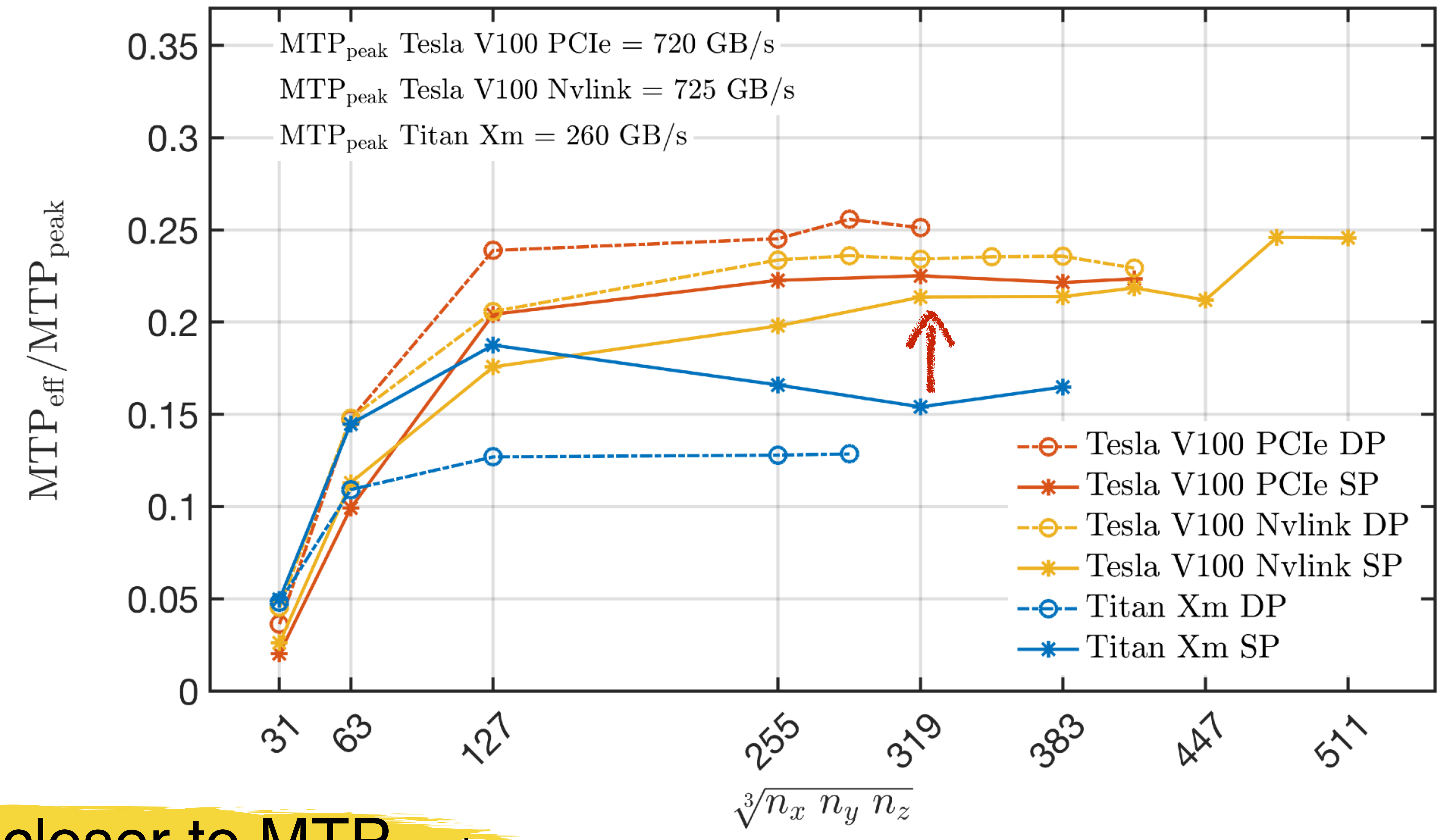
- Minimal # of memory access / iteration: 15 read + write (30 accesses)
- MTP_{peak} = memcpy only
- $MTP_{eff} < MTP_{peak}$: no neighbours read or access (derivatives) counted but they occur - (a stencil code).



Effective memory throughput

Effective and absolute metric to measure optimality of data access: $\text{MTP}_{\text{effective}}$

- Minimal # of memory access / iteration: 15 read + write (30 accesses)
- MTP_{peak} = memcpy only
- $\text{MTP}_{\text{eff}} < \text{MTP}_{\text{peak}}$: no neighbours read or access (derivatives) counted but they occur - (a stencil code).

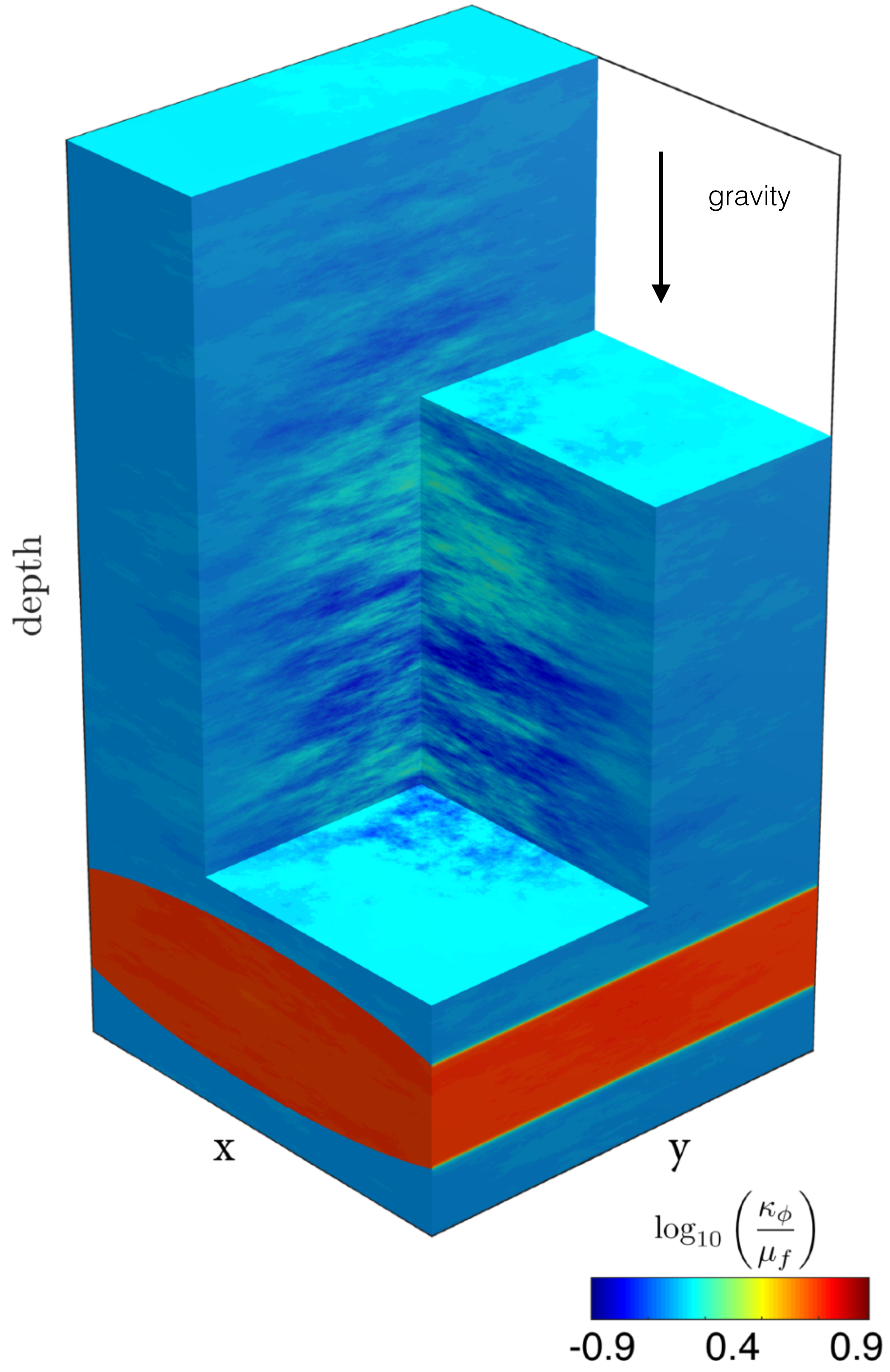


Future goes closer to MTP_{peak}

3-D fully coupled hydro-mechanics

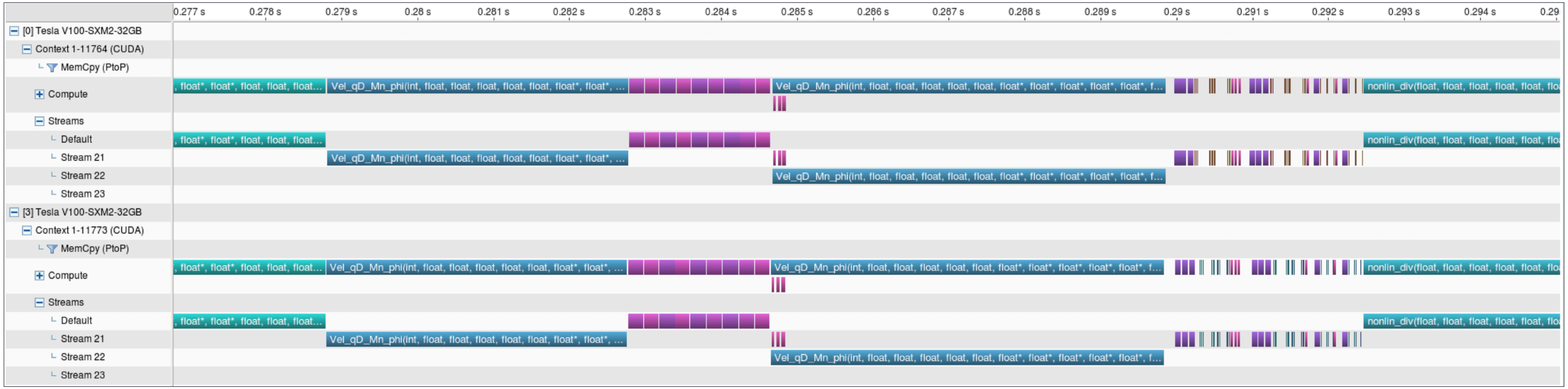
Spontaneous formation: 3-D numerical simulation

- Sediment deformation coupled to fluid motion
- Buoyancy driven flow: $\rho^{\mathrm{s}} = 2\rho^{\mathrm{f}}$
- Nonlinear mechanics (tensor + vector fields)
- Nonlinear fluid flow (vector field)
- Very high resolution:
  - space: 1000 x 1000 x 2000 grid points in 3-D
  - time: 20’000 time steps



Hiding communication

- Wrong implementation of CudaStream with blocking call to cudaDeviceSynchronize();



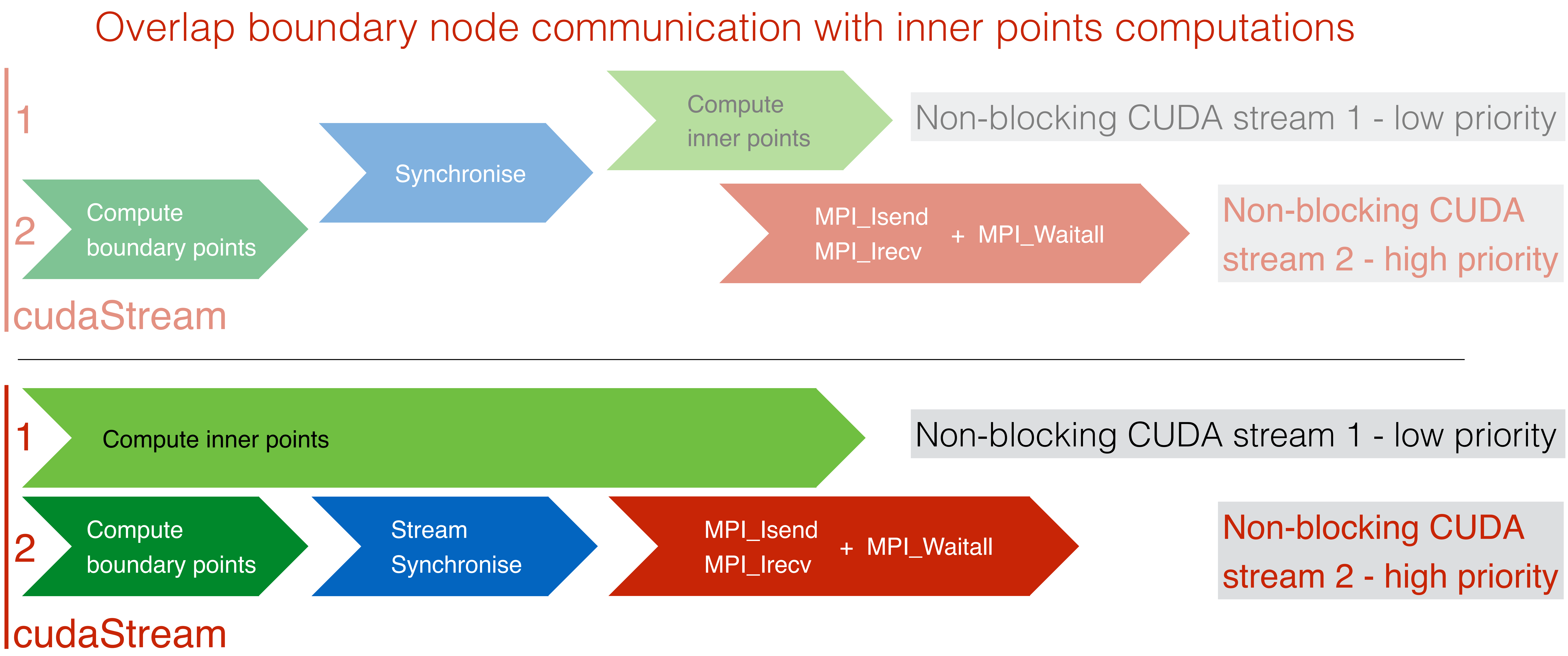
cudaStream implementation

CudaStream: simple approach to access memory on GPU while computations are ongoing:

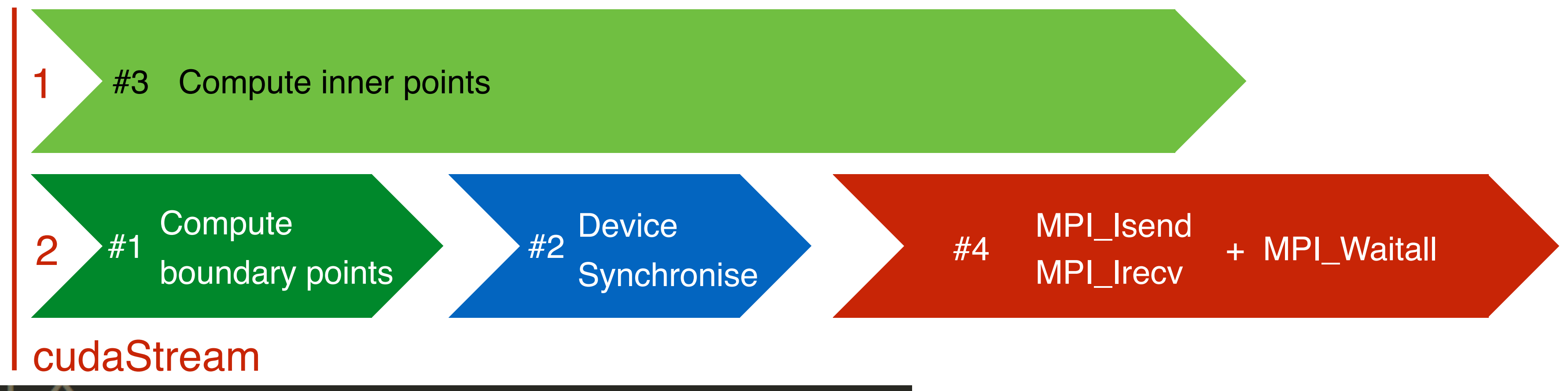
- Non blocking data accesses on the GPU as they happen on different non-blocking streams > flag for stream creation: `cudaStreamNonBlocking`.
- Syncing [Update boundaries] kernel does not interfere with [Compute inner points] kernel as it is done for stream 2 only: `cudaStreamSynchronize()`.
- Higher priority of stream 2 vs stream 1: `cudaStreamCreateWithPriority()`.
- Boundary width > 2 ensures optimal performance.

Hiding communication 2

- Alternative approach



Minimal changes to the code 2




```
#define BOUNDARY_WIDTH_X 32
#define BOUNDARY_WIDTH_Y 32
#define BOUNDARY_WIDTH_Z 32

__global__ void compute(int istep){
    int ix = blockIdx.x*blockDim.x + threadIdx.x; // thread ID, dimension x
    int iy = blockIdx.y*blockDim.y + threadIdx.y; // thread ID, dimension y
    int iz = blockIdx.z*blockDim.z + threadIdx.z; // thread ID, dimension z
    // Return if no work to do.
    if (   istep==0 &&     ix>=BOUNDARY_WIDTH_X && ix<=nx-1-BOUNDARY_WIDTH_X &&
                           iy>=BOUNDARY_WIDTH_Y && iy<=ny-1-BOUNDARY_WIDTH_Y &&
                           iz>=BOUNDARY_WIDTH_Z && iz<=nz-1-BOUNDARY_WIDTH_Z    ) return;
    if (   istep==1 && (   ix< BOUNDARY_WIDTH_X || ix> nx-1-BOUNDARY_WIDTH_X ||
                           iy< BOUNDARY_WIDTH_Y || iy> ny-1-BOUNDARY_WIDTH_Y ||
                           iz< BOUNDARY_WIDTH_Z || iz> nz-1-BOUNDARY_WIDTH_Z   ) ) return;
    //(...) (rest of the kernel as before)
}

main(){
    //(...)
    for (int istep=0; istep<2; istep++){
        compute<<<grid,block,0,streams[istep]>>>(istep);     // step 1 (istep==0) and step 3 (istep==1)
        if (istep==0) cudaStreamSynchronize(stream[0]);      // step 2
    }
    update_boundaries();                                      // step 4
    //(...)
}
```

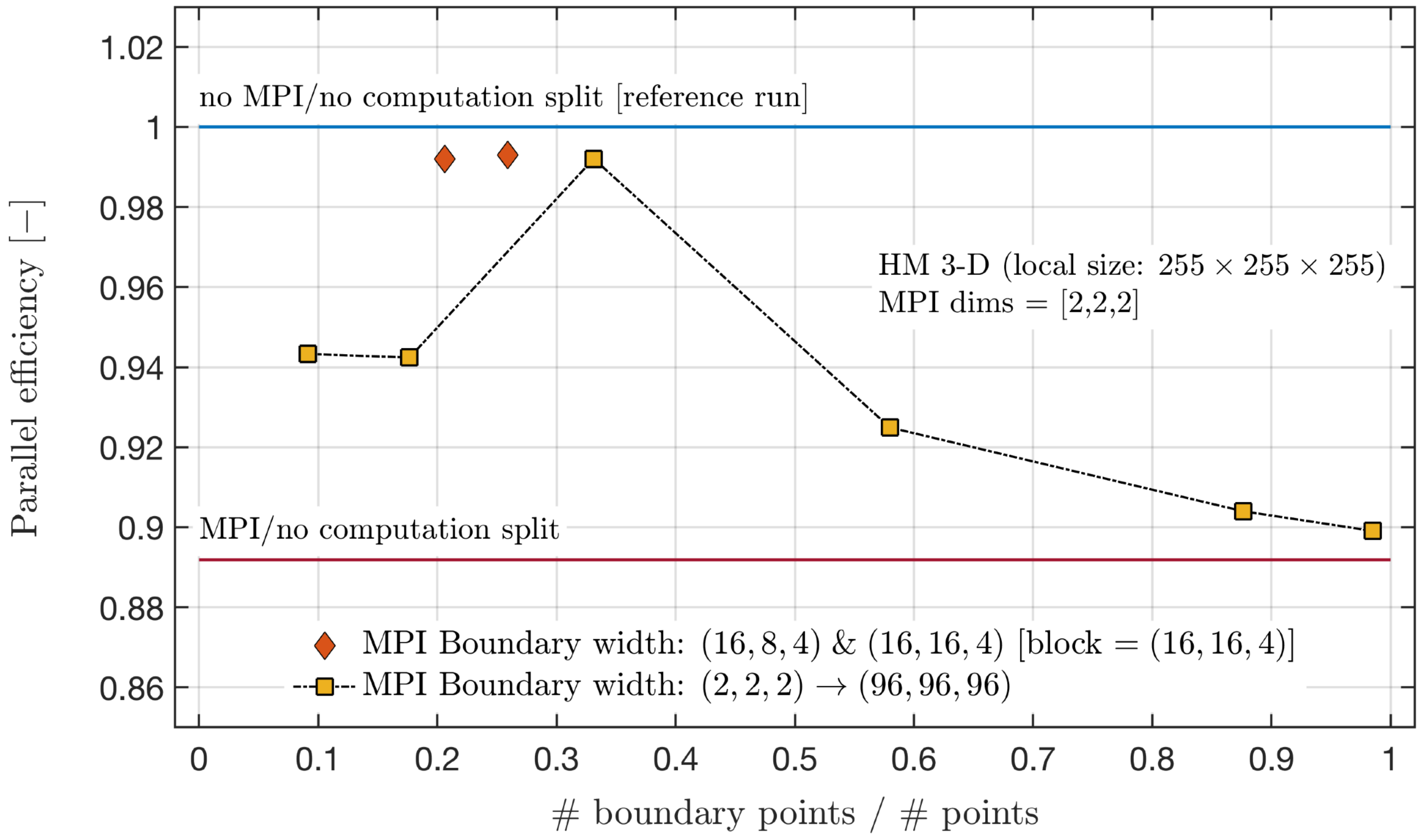
Hiding communication 2

Alternative approach for hiding MPI communication

- Synchronise only stream 2 (`cudaStreamSynchronize` instead of `cudaDeviceSynchronize`): only 1 modification to switch between two approaches !
- [compute inner points] begins simultaneous with [compute boundary points] kernels
- (+) No performance lost: boundary points & inner points are computed at the same time
- (-) Strongly rely on the stream priority feature to work very well
- (-) The less the priority is respected, the more the MPI data transfer begin will be delayed
- (-) Difficult to predict the time available for the MPI update

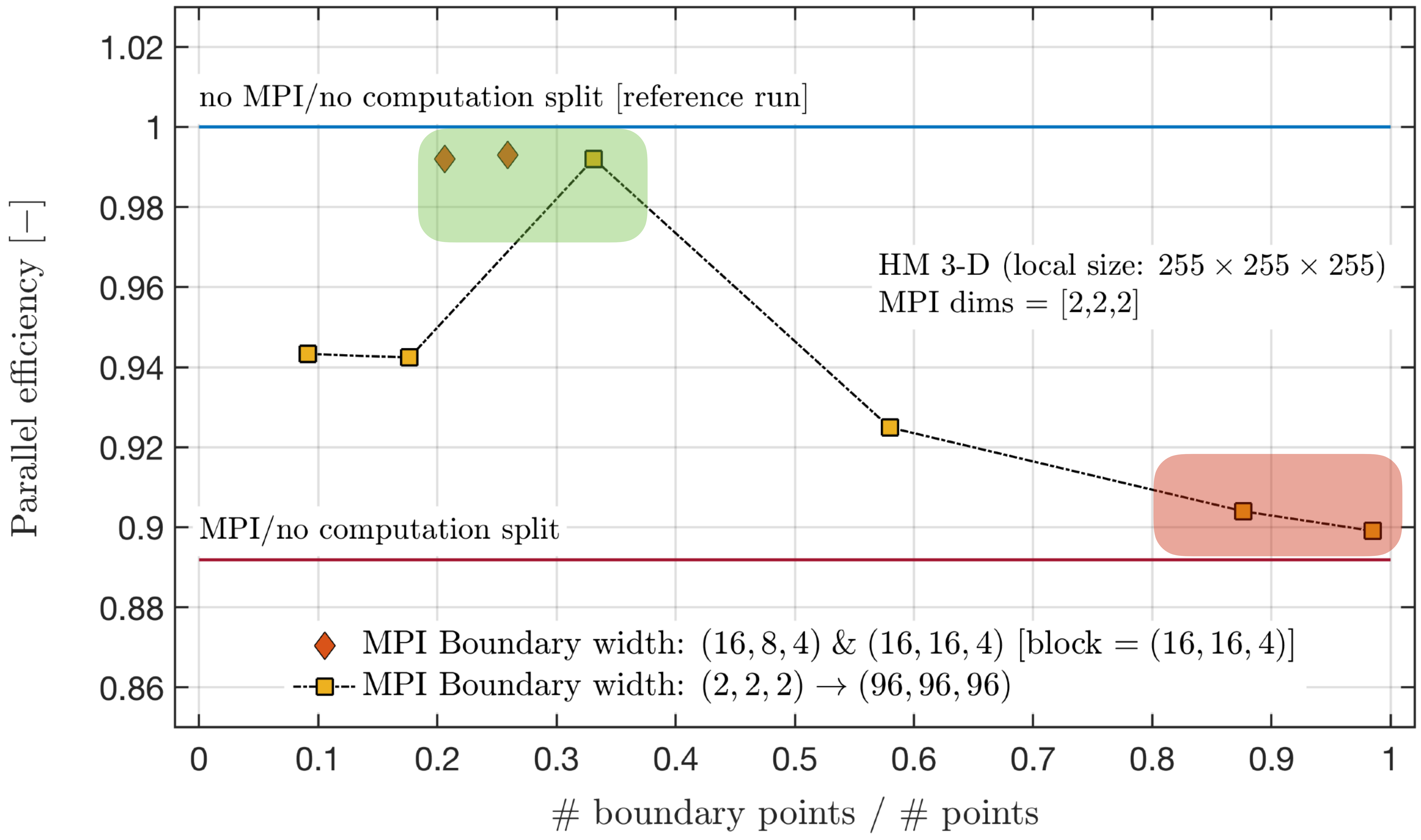
Avoid loss of performance

- Impact on the parallel efficiency | similar trend



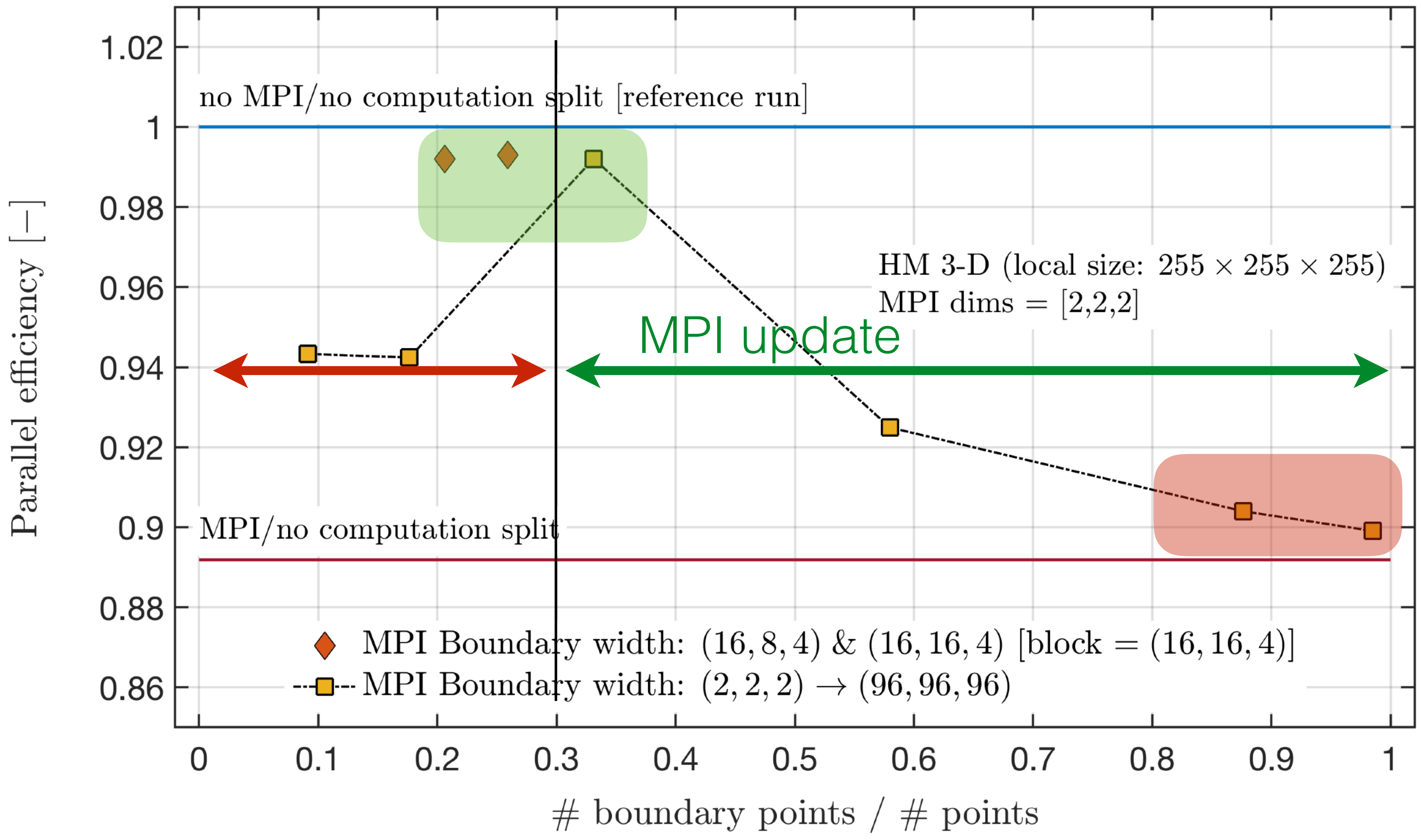
Avoid loss of performance

- Impact on the parallel efficiency | similar trend



Avoid loss of performance

- Impact on the parallel efficiency | similar trend



Optimality of data access

Optimality of data access

- PDE solvers are memory bound: computations $ | memory accesses $$$$
- Choose an appropriate solution strategy (solver):
  - Low memory footprint algorithm
  - Simple and regular data access pattern
- Computations are for free: recompute fields instead of storing them

/!\ High $MTP_{eff} \neq$ fast convergence (low # of iterations)

Performance limiters

- 15 fields to update R+W
- Bytes ≠ numbers (here float = 4 bytes)
- Current = main memory + cache
- Upper bound = minimal n_{IO}
- ratio = flops/bytes

0.1 < 1.8 << machine balances (17 or 24)

We are memory bound

3-D hydro-mechanics

Operation	Count	# flops	# bytes (float)	ratio
R+W	15	0	120	
$\frac{\partial}{\partial x}$	31	62	372	
Total (current)		62	524	0.1

GPU	# flops	# bytes (float)	ratio
Tesla V100	15.7×10^{12}	0.9×10^{12}	17
Titan Xm	12.1×10^{12}	0.5×10^{12}	24

Effective memory throughput

- Use an effective and absolute metric to measure optimality of data access: $\mathrm{MTP_{eff}}$
- Minimal # of memory transfers / iteration: $n_{\mathrm{RW}} = 30$ (R+W every 15 variable once)
- Tells us how far we are from ideal: compare to $\mathrm{MTP_{peak}}$ (memcopy only - no flops)

$$\boxed{\mathrm{MTP_{eff}} = \frac{n_{\mathrm{RW}}\, n_i^{\mathrm{tot}}\, n_{\mathrm{precis}}}{2^{30}\, t_{\mathrm{elapsed}}} \quad [\mathrm{GB/s}]}$$

$$\begin{aligned} n_{\mathrm{RW}} &= 2 \times (\text{read and write}) + \text{read only fields} \\ n_i^{\mathrm{tot}} &= n_x \times n_y \times n_z \times n_{\mathrm{t}} \\ n_{\mathrm{precis}} &= \text{word size [bytes]} \\ t_{\mathrm{elapsed}} &= \text{elapsed time [sec]} \end{aligned}$$

$\mathrm{MTP_{eff}}$ = lower bound of required memory transfers / time per iteration

$\neq$

$\mathrm{MTP_{profiler}}$ = performed memory transfers / time per iteration